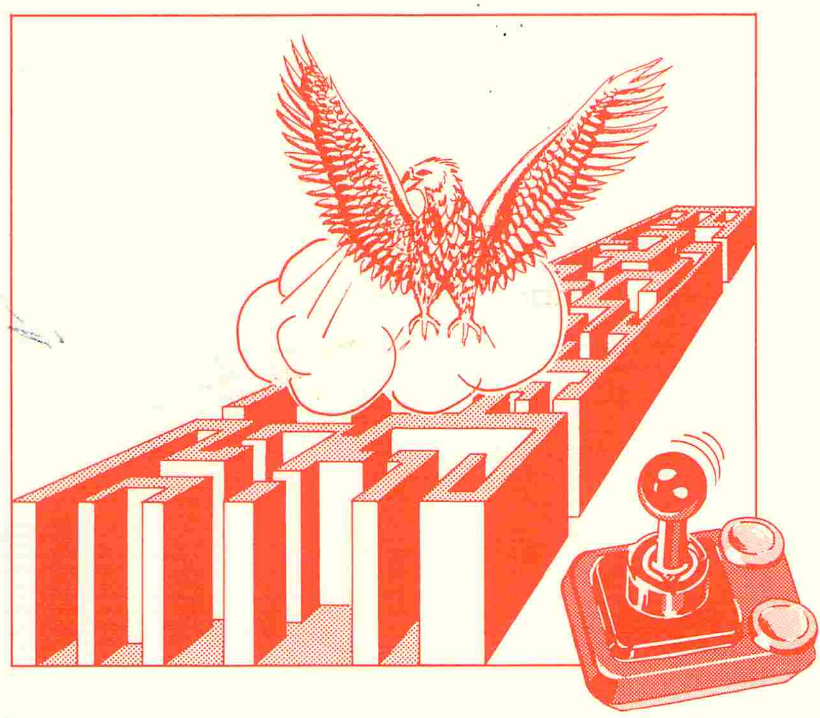
Linden

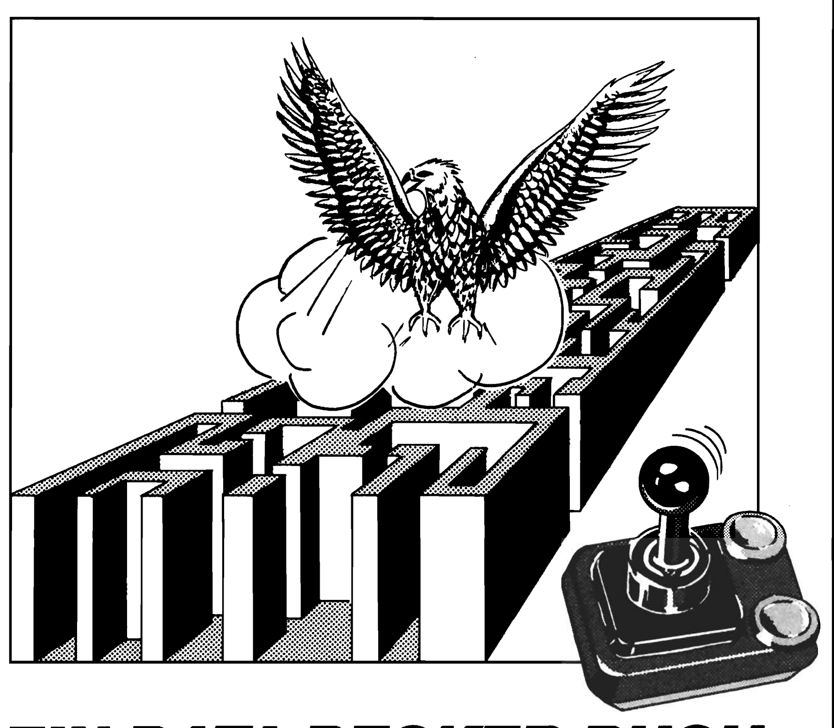
C 64 Superspiele selbstgemacht



EIN DATA BECKER BUCH

Linden

C 64 Superspiele selbstgemacht



EIN DATA BECKER BUCH

ISBN 3-89011-087-8

2. Auflage

Vorwort

Es sieht so einfach aus, wenn Pac Man durchs Labyrinth huscht oder Captain Future auf seinem Weg durch neue Galaxien die Feinde gleich dutzendweise zur Strecke bringt ...

... aber haben Sie als Homecomputer-Besitzer schon einmal versucht, ein Video-Spiel zu programmieren und sind dabei auf ungeahnte Schwierigkeiten gestoßen? Dann sind Sie mit diesem Buch gut beraten, denn es wird Ihnen mit Rat und Tat hilfreich zur Seite stehen, bei allen Problemen, die beim Programmieren von Video-Spielen auftreten können. Mit einer guten Anleitung gerät die Spiele-Programmierung sicher nicht zum frustrierenden Unterfangen, das erfahrungsgemäß damit endet, daß der Computer in eine Ecke verbannt wird.

Damit die Beschäftigung mit diesem Buch zum Erfolg führt, wird in abgestuften Lernschritten vorgegangen, um auch Interessierten, die nur wenig Praxis beim Erstellen von Programmen besitzen, ein sinnvolles Arbeiten zu ermöglichen. Bitte machen Sie sich die Mühe, alle Kapitel durchzuarbeiten, da sie aufeinander aufbauen.

Ich wünsche Ihnen viel Spaß bei der Bearbeitung dieses Buches und besonders bei der Entwicklung eigener Spiele.

Rüdiger Linden
Münster, im Juni 1985

I N H A L T S V E R Z E I C H N I S

Einleitung 1 1

TEIL 1

1.1 Die CPU 3
1.2 Das ROM 4
1.2.1 Das Betriebssystem 4
1.2.2 Der (BASIC-) Interpreter 4
1.2.3 Der Interrupt 5
1.3 Das RAM 6
1.3.1 Die Speicherverteilung des C-64 7
1.3.2 Die Zero-Page 8
1.3.3 Das Byte 1 9

2.1 Speicher schützen 11
2.2 Abspeichern geschützter RAM-Bereichen 14
2.3 Programme verketten 16

TEIL 2

3.1 Die Grafik 19
3.2 Die Blockgrafik 19
3.3 Hochauflösende Grafik 20
3.3.1 Einschalten der hochauflösenden Grafik 23
3.4 Blockgrafik mit selbstdefinierten Zeichen 26
3.4.1 Extended-Color-Modus 26
3.4.2 Multi-Color-Modus 27
3.5 Zeichen selbst definieren 29

4.	Sprite-Grafik	34
4.1	Sprites einschalten	34
4.2	Vergrößern von Sprites	35
4.3	Farbgebung	36
4.4	Multi-Color-Sprites	37
4.5	Priorität	38
4.6	Sprites bewegen	40
4.7	Veränderung der Form	41
4.8	Kollisionen	43
4.8.1	Sprite-Sprite-Kollisionen	43
4.8.2	Sprite-Hintergrund-Kollisionen	43
5.	Sound-Programmierung	45
5.1	Der SID	45
5.2	Lautstärke	45
5.3	Die Hüllkurve	46
5.4	Das Tastverhältnis	47
5.5	Die Frequenz	47
5.6	Die Wellenform	48
6.	Der Joystick	51
7.	Das Hexadezimalsystem	53
8.	Das Binärsystem	55
9.	Binäre Arithmetik	58
9.1	AND	58
9.2	OR	59
9.3	NOT	60
9.4	XOR (EOR)	61
10.	Basic kontra Maschinensprache	63
10.1	Was ist "Maschinensprache"?	64

11. Maschinensprache-Einführung 67
11.1 Wie funktioniert eigentlich ein Computer? 67
11.2 Abschied von den Variablen 68
11.3 Liste der Maschinensprachebefehle 70

12. Interrupt-Erweiterungen 81
12.1 Veränderung des Interrupts 82
12.2 Veränderung des Interrupt-Vektors 83

13. Tele-Spiele 85

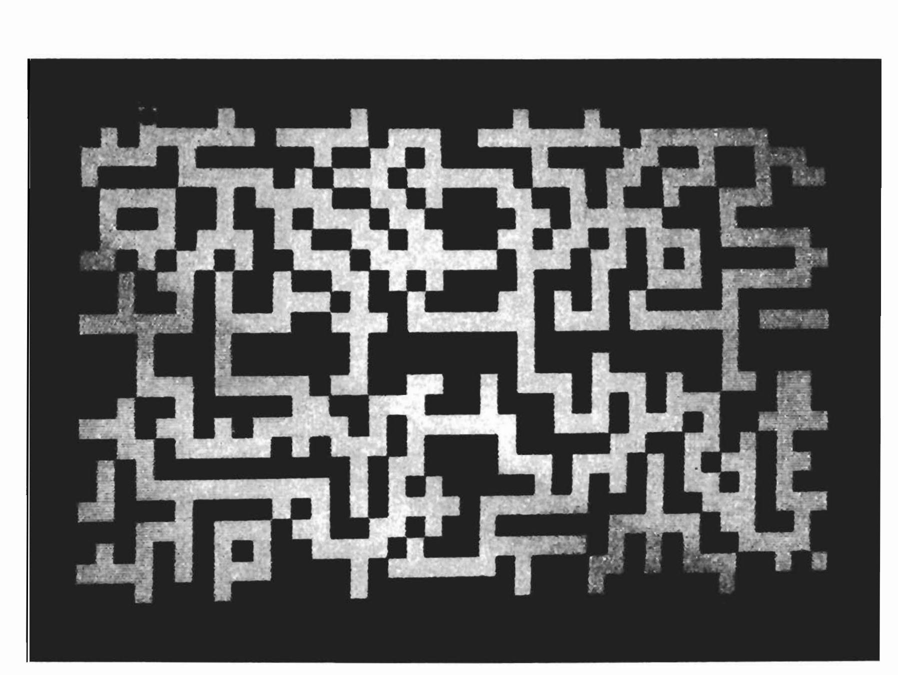
14. Basic für Telespiele 89

TEIL 3

14.1 Die Spielidee 95
14.2 Überprüfung auf Durchführbarkeit 98
14.3 Überwachung der Spielzustände 99
14.4 Die Programmierung 102
14.5 Beispielprogramme 103
14.5.1 Labyrinth 103
14.5.2 Car 113
14.5.3 3-D-Autorennen 121
14.5.4 Bird 176

TEIL 4

15.1 Sprites über den Bildschirm bewegen 225
15.2 Sprite-Editor 227
15.3 DATA-Erzeuger 233
15.4 DEZ in N 234



Einleitung

Das Buch ist gegliedert in vier Teile, deren Thematik insgesamt auf unser Ziel hinarbeitet.

Teil 1 gibt eine Einführung in den inneren Aufbau des C-64. Hier finden Sie Informationen über die allgemeine Konstruktion von Computern, den Speicherbelegungsplan des C-64, die Funktion von ROM und RAM und einige Tips, wie Sie die durch die Software bedingten Grenzen des Basic V2.0 in den für uns wichtigen Gebieten umgehen können.

Teil 2 wird uns mit den Sachen vertraut machen, die wir beherrschen müssen, um effektiv Spiele schreiben zu können. Hier werden dann auch Dinge angesprochen, die in der Anleitung zum 64er einfach zu kurz kommen.

Teil 3 beschäftigt sich mit der Programmierung von Spielen. Es wird gezeigt, welche Techniken nötig sind, um Spiele der verschiedensten Arten zu programmieren. Hier wird es dann allerdings auch ein wenig schwieriger. Werden wir doch auf eine neue Art zu programmieren umsteigen, der Programmierung in Maschinensprache. Doch keine Angst: Auch diejenigen, die sich zu einem solchen Schritt nicht entschließen können, werden ihren Nutzen aus diesem Teil ziehen können. Es werden einige nützliche Routinen vorgestellt werden, die man übernehmen und verwenden kann, die man aber nicht unbedingt verstehen muß.

Teil 4 stellt drei Hilfsprogramme vor, mit denen ich Ihnen die Arbeit des Programmierens erleichtern möchte.

Noch eine Anmerkung: In diesem Buch steht das Zeichen ^ für den ↑.

TEIL 1

Um mit unserem Computer effektiv arbeiten zu können, muß man sich ein wenig mit dem internen Aufbau beschäftigen. Zu diesem Zweck möchte ich Sie jetzt zu einer kleinen Rundreise durch unseren C-64 einladen. Keine Angst, ich beabsichtige nicht, einem Informatik-Studium vorzugreifen, sondern werde mich nur zu dem Wesentlichen äußern.

1.1. Die CPU

Die CPU oder, mit anderem Namen, der Microprozessor ist das Herz eines Computers. Man kann ihn mit einem König in seinem Reich vergleichen. Er ist der große Koordinator, dem alle anderen Bauteile mehr oder weniger untergeben sind. Um seine Regierungsfunktion ausüben zu können, verfügt er über verschiedene Leitungen (oder Busse), mit denen er sein Reich kontrolliert. Da wäre zum einen der "16 Bit breite" Adressbus. 16 Bit breit bedeutet nichts anderes, als daß dieser Bus über 16 Leitungen verfügt. Mit diesen Leitungen kann die CPU auf 2^16=65536 Adressen zurückgreifen. Aus diesen Adressen kann er dann mit Hilfe eines anderen Busses Daten lesen oder aber dort Daten ablegen. Dieser Datenbus hat bei unserem Prozessor (dem 6510) acht Leitungen, ist also ein Acht-Bit-Bus. Damit ist festgelegt, daß jedes Datum eine aus 8 Bit zusammengesetzte Information ist (das entspricht einer Zahl zwischen 0 und 255). Aus dieser Tatsache resultiert auch die Einordnung des 6510 in die Gruppe der 8-Bit-Prozessoren.

1.2. Das ROM

Das ROM (Read Only Memory, zu deutsch: Festwertspeicher) liefert der CPU alle Informationen, die der Rechner vom Moment des Einschaltens an benötigt, um z.B. die Computersprache Basic zu verstehen, aber auch Routinen, die dem Benutzer erst gar nicht bewußt werden. Um das ROM ein wenig zu beleuchten, wollen wir es erst mal in die beiden wichtigsten Teile aufspalten.

1.2.1. Das Betriebssystem

Das Betriebssystem regelt die Kommunikation des Benutzers mit dem Computer, darüber hinaus aber auch das Zusammenspiel von Computer und Peripheriegeräten. Eine wichtige Aufgabe ist z.B. die Abfrage der Tastatur oder der Aufbau des Bildschirms.

1.2.2. Der (Basic-) Interpreter

Der Interpreter (engl. für Übersetzer) hat zum einen die Aufgabe, alles Eingetippte auf Verwendbarkeit zu überprüfen, zum anderen, die Befehle in eine Form zu bringen, so daß die CPU sie bearbeiten kann. So wird z.B. alles Eingegebene erst mal auf die Syntax hin überprüft. Entspricht diese nicht den eng gesetzten Grenzen, so sieht sich der Interpreter dazu genötigt, der Sache mit einer Fehlermeldung Einhalt zu gebieten. Ist diese Klippe aber umschifft, so beginnt erst die richtige Arbeit für den Interpreter.

Als erstes wird unterschieden, ob die Eingabe im Direkt- oder im Programmodus erfolgt ist. Danach erfolgt entweder

die direkte Abarbeitung, d.h. der Interpreter macht die CPU darauf aufmerksam, an welcher Stelle die Routinen stehen, die abzuarbeiten sind, oder im anderen Fall wird die Eingabe umgewandelt und in den Basic-Speicher geschrieben. Bei dieser Umwandlung werden z.B. die Basic-Befehle bereits in sogenannte Tokens übersetzt, um so die Befehle bei dem eigentlichen Ablauf des Basic-Programms schneller übersetzen zu können.
Dies klingt alles so, als wäre der Interpreter gewissermaßen ein Staat im Staate, in Wirklichkeit ist aber auch er nur eine Routine, die von der CPU ständig bearbeitet wird.

1.2.3. Der Interrupt

Wir wissen also jetzt, daß mit dem Einschalten des Computers zwei Maschinensprache-Programme gestartet werden, ohne die ein Arbeiten mit dem Computer unmöglich wäre. Es ist aber auch klar, daß, falls eines der beiden Programme fehlen würde, der Anwender nur einen geringen Nutzen durch das andere Programm hätte. Das bedeutet, daß für einen reibungslosen Ablauf beide Programme gleichzeitig laufen müßten. Mit dieser Forderung ist unser Prozessor aber auch schon völlig überlastet. Er ist nur in der Lage, ein einzelnes Programm zu bearbeiten. Um trotzdem über beide Programme "ständig" verfügen zu können, haben sich die Ingenieure von Computern etwas einfallen lassen: Den Interrupt. Diese Routine sorgt dafür, daß alle 60stel Sekunde die CPU aus ihrer Arbeit gerissen wird, um sich mit dem Betriebssystem zu beschäftigen. Ist diese Arbeit geschehen, so rechnet die CPU an der Stelle des ursprünglichen Programms weiter, an der sie durch den Interrupt herausgerissen wurde.

Auf diese Art ist gewährleistet, daß die beiden ROM-Programme, Interpreter und Betriebssystem, quasi gleichzeitig arbeiten. Das Fachwort für die Technik des

gegenseitigen Aufrufes lautet Time-sharing. Nach diesem Prinzip laufen sämtliche Großanlagen, die z.B. über mehrere Benutzerplätze verfügen.

Nicht nur mit dieser Pflicht leistet der Interrupt für uns wertvolle Arbeit. Man kann ihn auch für andere Dinge gebrauchen und gerade für unser Ziel, die Spieleprogrammierung, sinnvoll einsetzen. Wir werden noch sehen, daß er uns dort eine Menge Arbeit abnehmen kann und auch wird.

Zusammenfassend kann man also sagen, daß unsere 20 K ROM (ein "K" entspricht einer Anzahl von 2^10 = 1024 Bytes), gefüllt mit Betriebssystem und Interpreter, es uns erst ermöglichen, mit unserem Computer zu arbeiten. (Die Tatsache, daß das ROM noch ein bißchen mehr beherbergt, soll uns vorerst nicht interessieren, ich werde die für uns interessanten Stellen im späteren Verlauf des Buches vorstellen.)

1.3. Das RAM

Das RAM (Random Access Memory oder Schreib/Lese-Speicher) ist der freie Speicherplatz, in den wir unsere Programme, Daten und anderes schreiben. Wie der Name schon sagt, verfügt der C-64 über 64 K = 65536 Bytes. Dies ist gleichzeitig der maximale Speicherraum, der von einer CPU wie der 6510 verwaltet werden kann. Hmm... wie wir gerade erfahren haben, kann also unser Prozessor 64 K verwalten. (Wir erinnern uns hoffentlich an den 16-Bit-Adress-Bus.) Wir wissen jetzt auch, daß wir über 64 K RAM verfügen. Hmm.... wo bleibt unser ROM? Und warum gibt uns der C-64 beim Einschalten bekannt, das wir nur über 38911 "Basic Bytes" verfügen? Die Lösung ist im sogenannten Bank-switching zu finden. Unser Computer ist in der Lage, bestimmte Adressbereiche doppelt zu nutzen. So gibt es zwar

tatsächlich 64 K RAM, allerdings sind gewisse Bereiche davon durch das ROM überlagert.

1.3.1. Die Speicherverteilung des C-64

Machen wir folgende Rechnung: Wir haben 64K RAM. Von diesen 64 K müssen wir erst einmal 20 K abziehen, weil dieser Bereich vom ROM überlagert ist, und wir an diesen nicht ganz so einfach heran kommen. Übrig bleiben 44 K. Davon müssen wir wiederum 4 K abziehen, die nur für Programmierung in Maschinensprache vorgesehen sind. Von den restlichen 40 K gehen noch 2 K für die Zero-Page und für das Video-RAM ab. Übrig bleiben 38 K; und das sind jene 38911 Bytes, die uns der C-64 beim Einschalten meldet.

Untergliedern wir nun einmal genauer:

Adresse:	Genutzt für:
0-1023	Zero-Page
1024-2047	Bildschirmspeicher
2048-40959	38 K Basic-Speicher
40960-49151	8 K Basic-Interpreter
49152-53247	4 K RAM (nur für Maschinensprache)
53248-57343	4 K Character-Generator
53248-57343	I/O-Bereich
57344-65535	8 K Betriebssystem

Dem aufmerksamen Leser wird aufgefallen sein, daß der Bereich von 53248 - 57343 doppelt vorkommt. Das liegt daran, daß an dieser Stelle nicht nur RAM und Char-ROM übereinander liegen, sondern noch ein weiterer Bereich, der sogenannte I/O- oder Input/Output-Bereich. In diesem Bereich liegen die Bausteine, die für die Tastatur verantwortlich sind, die das Fernsehbild aufbauen oder die den Ton produzieren. Hier liegt dann auch das Color-RAM (von 55296 - 562959), 1000 Bytes, die für die Farbe der Zeichen auf dem Bildschirm verantwortlich sind.

1.3.2. Die Zero-Page

Im eigentlichen Sinne versteht man unter der Zero-Page die ersten 256 Bytes, die erste Speicherseite. Beim C-64 kann man aber ohne weiteres sagen, daß sie das erste Kilobyte einnimmt. Wenn Sie in Ihrem Handbuch einmal im Anhang Q herumblättern, dann werden Sie schnell feststellen, was es mit der Zero-Page so auf sich hat. Sie ist eine Ansammlung von wichtigen und wichtigsten Speicherzellen, ohne die Ihr C-64 ziemlich ratlos in der Gegend stehen würde.

Nehmen wir einmal das Handbuch zur Hand und blättern den Anhang Q auf. Beim Durchlesen wird uns auffallen, daß immer wieder von Zeigern, Vektoren und Pointern die Rede ist. Dies sind Bezeichnungen für Adressen, zu denen die CPU "springen" muß, um gewisse Routinen wie z.B. die Interrupt-Routine zu finden. Die Zeiger haben eine Länge von 2 Bytes. Die Adresse, auf die sie zeigen, läßt sich wie folgt berechnen: 1. Byte + 256 * 2. Byte = Adresse. Das erste Byte eines Zeigers nennt man auch Low-Byte, das zweite dementsprechend auch High-Byte. Diese Form der Abspeicherung gilt im übrigen

für fast alle Zwecke, bei denen eine Adresse vonnöten ist: erst kommt das Low-Byte, dann das High-Byte.

Wir können einmal probeweise einen solchen Pointer verändern. Aber bitte nur, wenn wirklich keine wichtigen Daten gespeichert sind. Probieren Sie einmal aus: poke 789,0 (Interupt-Vektor).

1.3.3. Das Byte 1

Die ganze Zeit ist die Rede von 64 K RAM, die vorhanden sind, aber nicht nutzbar, da vom ROM überlagert. Was hat es damit auf sich? Sind die restlichen 20 K RAM nur eine kostspielige Spielerei der Commodore Ingenieure, da man offensichtlich doch nicht in den Genuß kommt, sie zu nutzen? Es muß doch einen Weg geben, diese für uns erreichbar zu machen. Ich kann Sie beruhigen, es gibt einen. Dazu aber noch folgendes: Man kann jederzeit durch POKEn in diese Bereiche Daten ablegen. Doch so ohne weiteres kommt man an diese Daten nicht mehr heran. Ein PEEKen von einer dieser Adressen liest nur das darüber gelagerte ROM aus.

Wie Sie sich schon richtig gedacht haben werden, so hat das irgend etwas mit der Überschrift zu tun. Das Byte 1 regelt, welche Teile des ROMs zur Zeit gültig sind bzw. an welchen Stellen das darunter liegende RAM "auftaucht".

Wichtig an diesem Byte sind nur die ersten 3 Bits. Im Normalfall sind sie alle drei gesetzt. Werden eines oder mehrere von ihnen gelöscht, so geschieht folgendes:

Bit 0:

Wird dieses Bit gelöscht, etwa mit POKE 1, PEEK(1) AND 254, so wird das Basic-ROM im Bereich von 40960 - 49151 abgeschaltet. Das darunter liegende RAM liegt theoretisch

frei. Theoretisch, weil wir nämlich Schwierigkeiten haben dürften, auf das RAM zurückzugreifen. Zur Erinnerung: Wir haben gerade unser Basic "rausgeschmissen". Ein Zugriff ist also nur über die Maschinensprache möglich.

Bit 1:

Auch hier wird das Basic-ROM abgeschaltet, zusätzlich jedoch auch das Betriebssystem. Man kann also beinahe sagen: Nichts geht mehr. ("Beinahe" deshalb, weil sich ein in Maschinensprache geschriebenes Programm durch derartige Nichtigkeiten erst gar nicht tangieren läßt.)

Ich glaube, Sie geben mir recht, wenn ich behaupte, 38 K RAM sind vorerst genug. Lassen wir also Bit 0 und 1 in Frieden und betrachten gemeinsam Bit 2.

Bit 2:

Dieses Bit entscheidet darüber, ob wir das Character-ROM auslesen können. Aber auch hier wird sich der Rechner kurzzeitig verabschieden, wenn wir das Bit kurzerhand löschen.

Der Grund? Mit dem Löschen von Bit 2 verzichten wir darauf, unseren I/O-Bereich anzusprechen... Tastaturabfrage geht also nicht (d.h. der Rechner versucht es natürlich, wird aber irgendwo in den Character-Generator springen und anschließend den Dienst verweigern). Trotzdem ist es möglich, das Character-ROM auszulesen und das sogar vom Basic her. Wie das möglich ist, davon später.

2.1. Speicher schützen

Für gewisse Situationen müssen wir in der Lage sein, gewisse Speicherbereiche vor dem Zugriff des Interpreters zu schützen.

Ein solcher Fall kann eintreten, wenn wir mehr als 4 Sprites in einem Programm verwenden wollen.

Wir können Speicherplatz schützen, indem wir dem Interpreter mitteilen, daß ein Basic-Programm statt an der gewohnten Stelle (2048) erst später anfängt. Der elektronische Briefkasten, wo wir eine solche Nachricht deponieren können, liegt in der Zero-Page (wo denn auch anders?). Es handelt sich um die Speicherzellen mit der Adresse 43 und 44.

Bevor wir uns an das Verlegen des Basic-Bereiches heranmachen, möchte ich noch auf eine Kleinigkeit aufmerksam machen: Der Pointer unter der Adresse 43/44 zeigt nicht auf den Basic-Start, sondern auf die erste Basic-Zeile. Der Unterschied ist nicht gravierend, aber doch von Bedeutung. Vor der ersten Basic-Zeile steht nämlich eine 0 im Speicher. Verschieben wir den Basic-Bereich, so müssen wir dafür sorgen, daß der Interpreter auch am neuen Ort eine Null vorfindet.

Kommen wir nun zum Verschieben und stellen uns die folgende Situation vor:

Wir haben ein Programm geschrieben, in dem wir 1 K geschütztes RAM benötigen (z.B. für 15 Sprites). Wir verlegen also das Basic von 2048 auf 3072, der Platz von 2048 - 3071 ist dann unser geschütztes RAM.

Schritt 1: Berechnung der Pointer

Wir wollen den Basic-Anfang auf 3072 verschieben. Da, wie

bereits gesagt, als erstes Zeichen im Basic-Bereich eine Null stehen muß, wird der Pointer so berechnet, daß er auf die Adresse 3073 zeigt.

Die Berechnung geht wie folgt vor sich:

Der Pointer spaltet sich in zwei Teile auf: Low-Byte und High-Byte.

High-Byte: INT (3073 : 256)

Low-Byte : 3073 - 256 * High-Byte

2. Schritt: Verändern des Pointers

Dies geschieht einfach durch POKEn der beiden Werte:

POKE 43, Low-Byte

POKE 44, High-Byte

3. Schritt: Die Null als erstes Byte im BASIC-RAM

Auch hier genügt ein einfacher POKE:

POKE 3072, 0

4. Schritt: Korrigieren der anderen Pointer

Man sollte nicht meinen, daß es der C-64 so einfach hinnimmt, wenn man versucht, seine innere Ordnung auf den Kopf zu stellen. Im Gegenteil, unser Rechner kann auf so etwas sehr "sauer" reagieren. Um dieses zu verhindern, genügen zwei Befehle zur Abhilfe:

NEW:CLR

Zum Schluß noch eine Information: Es wird bei der oben

beschriebenen Technik lediglich der Start des Basic-RAMs verschoben, nicht jedoch ein eventuell vorhandenes Programm. Dieses verabschiedet sich bei der Prozedur. Deshalb: Vorsicht bei Manipulationen an der Zero-Page. Wichtige Daten und Programme vorher abspeichern (nach dem Verschieben kann man sie auf gewohnte Art mit LOAD "Filename", 8 (1) wieder einladen).

2.2. Abspeichern geschützter RAM-Bereiche

Die schönste Graphik nützt uns nichts, sind wir nicht in der Lage, sie irgendwie auf Dauer zu sichern, daß heißt abzuspeichern. Hierbei gibt es mal wieder ein Problem. Unsere SAVE-Routine ist nämlich bestenfalls eine Minimal-Lösung. Will man z.B. den Bildschirm-Inhalt auf Diskette oder Cassette abspeichern, so muß man einen anderen Weg gehen, da der SAVE-Befehl lediglich für Programme gedacht ist. Aber auch hier gibt es einen Weg: Die Lösung liegt, natürlich möchte man fast sagen, wieder einmal in der Zero-Page.

Ich möchte an dieser Stelle einfach je eine Möglichkeit angeben, wie es möglich ist, solche Bereiche abzuspeichern, da alles weitere den Rahmen dieses Buches sprengen würde. Denjenigen, der sich darüber hinaus für solche Techniken interessiert, möchte ich auf die ebenfalls bei DATA BECKER erschienene Literatur hinweisen (PEEKS & POKES o.ä.).

Cassetten-Version

```
10 REM FILENAME
20 POKE 193, SL: POKE 194, SH : REM Startadresse (Low/High)
30 POKE 174, EL: POKE 175, EH : REM Endadresse (Low/High)40
POKE 187, PEEK (43) + 6: POKE 188, PEEK (44) : REM Zeiger
auf Filename
50 POKE 183, L : REM Filenamenlänge
60 POKE 186, 1: POKE 185, 0 : REM Gerät/Sekundäradresse
70 SYS 62954 : REM Aufruf der SAVE Routine im ROM
```

Zur Erläuterung dieses Programms:

Hinter das REM in Zeile 10 schreiben wir den Namen, den das Programm bekommen soll. In Zeile 40 geben wir dem

Betriebssystem bekannt, wo dieser Name zu finden ist. Hier kann es gelegentlich zu Problemen kommen, wenn nämlich PEEK (43) + 6 einen größeren Wert als 255 einnimmt. Ist dies der Fall, müssen wir den Pointer in Zeile 40 auf herkömmliche Weise ausrechnen.

Zeile 50: L gibt die Länge des Filenamens an.

Zeile 20: SL/SH ist ein Pointer auf den Anfang des abzuspeichernden Bereich, wir müssen ihn vor dem Ablauf dieses Programms per Hand ausrechnen.

Zeile 30: EL/EH ist ebenfalls ein Pointer, der vorher berechnet werden muß. Er zeigt auf das Ende des Programms.

Etwas einfacher haben es hier die Leute, die über eine Disketten-Station verfügen. Hier die für sie geeignete Version:

```
10 open 1,8,1,"0:Filename"20 PRINT #1, CHR$ (0);: PRINT #1
CHR$ (4);: REM Startpointer
30 FOR I = 1024 TO 2023
40 PRINT #1, CHR$ (PEEK (I));
50 NEXT I: CLOSE 1
```

Zeile 10 erzeugt ein Directory mit Namen Filename.

Zeile 20: Hier wird auf der Diskette abgespeichert, an welcher Stelle das Programm anfängt. Es handelt sich hier um einen Pointer im üblichen Sinne. Er muß auf den Anfang des zu speichernden Bereich zeigen.

Die Schleife in Zeile 30 - 50 liest immer ein Zeichen aus dem Bereich und gibt dieses an die Diskette weiter. Das CLOSE 1 schließt das in der Zeile 10 geöffnete File. Fehlt es, so würde das File auf der Diskette zerstört werden.

Diese beiden Routinen sind übrigens, mit freundlicher Genehmigung,dembeiDATABECKERerschienenenBuch"PEEKS& POKES zum Commodore 64" entnommen.

2.3. Programme verketten

Es ist wohl jedem von uns schon einmal passiert, daß er aus zwei verschiedenen Programmen ein drittes, neues machen wollte. Für einen solchen Fall ist unser Basic aber wieder einmal ungeeignet. Es fehlt der Befehl MERGE.

Trotzdem gibt es eine Möglichkeit, mehrere Programmteile zusammenzufügen, und das im Direktmodus, d.h. ohne jedes Hilfsprogramm.

Sie ahnen es schon, die Zero-Page leistet uns auch hier wertvolle Dienste.

Der zu beschreitende Weg ist folgender: Wir laden den ersten Teil der zu verknüpfenden Programme ein und ... schützen ihn einfach, indem wir die entsprechenden Pointer verändern. Danach können wir den zweiten Teil einladen, ohne daß dem Programm etwas passiert. Als letztes setzen wir die Pointer wieder zurück und verknüpfen auf diese Weise die Programme.

Beachten sollte man allerdings, daß der zweite Teil über höhere Zeilennummern verfügt als der erste; der Interpreter könnte sonst nämlich gehörig ins Stolpern kommen.

1.Schritt: Notieren der alten Pointer

Notieren Sie sich die ursprünglichen Werte der Adressen 43 und 44. Wir brauchen sie, um das endgültige Programm erscheinen zu lassen.

2.Schritt: Schützen des Programms

Wie wir den Anfang eines Programms erfahren, wissen wir bereits. Das Ende erfahren wir wie folgt. Unter den Adressen 45/46 ist ein Pointer abgespeichert, der auf den Anfang des Variablen-Bereiches zeigt. Dieser wiederum beginnt zwei Bytes hinter unserem Basic-Programm. Wir lesen also diese beiden Adressen aus, berechnen daraus den Variablen-Start, ziehen von dieser Adresse 2 ab und berechnen daraus den neuen Pointer. POKEn Sie jetzt den Pointer in die Adresse 43/44.

3.Schritt: Initialisieren der übrigen Pointer

Geben Sie NEW ein und die Pointer sind initialisiert. Dem Programm geschieht nichts, es befindet sich ja in einem geschützten Bereich.

4.Schritt: Laden des zweiten Programmteils

Laden Sie jetzt den zweiten Teil des Programms. Sie dürfen dabei aber auf keinen Fall mit LOAD "Name",X,1 arbeiten, weil sich das Programm in diesem Fall nicht an die Pointer halten würde, sondern an die Information, die beim SAVEn mit abgespeichert wurde; unser Programm würde zerstört werden. Verwenden Sie also nur den Befehl LOAD "Name",X.

5.Schritt: Das Verketten

POKEn Sie an die Adresse 43/44 den Wert, den Sie sich zu Beginn der Prozedur hoffentlich gemerkt haben. Damit wäre die Arbeit getan, die Programme sind verkettet.

TEIL 2

3.1. Die Graphik

Die Graphik ist das wichtigste Gestaltungsmittel für Spiele. Je nachdem wie aufwendig diese programmiert ist, desto mehr Spaß wird das fertige Spiel hinterher bereiten. Der C-64 bietet verschiedene Möglichkeiten, Graphik auf den Bildschirm zu bringen. Die einfachste Technik ist die Blockgraphik unter Verwendung der vom Rechner vorgegebenen Zeichen. Die aufwendigste Technik ist die sogenannte hochauflösende Graphik mit insgesamt 64000 einzeln programmierbaren Punkten auf dem Bildschirm. Mit ihr kann man ganz hervorragende Ergebnisse erzielen, muß diese jedoch durch einen immensen Aufwand erkaufen. Die dritte Technik und gleichzeitig die für uns interessanteste ist eine Blockgraphik mit selbst definierten Zeichen. Mit ihr kann man ähnlich gute Resultate erzielen, wie mit der hochauflösenden Graphik, jedoch mit bedeutend weniger Aufwand.

3.2. Die Blockgraphik

Eine solche Graphik zu erstellen, ist nicht weiter schwer. Man benutzt einfach die Buchstaben, Graphikzeichen etc., die von vornherein durch das ROM gegeben sind. Man erreicht diese Zeichen entweder in Verbindung mit der (Shift)- oder der (C=)-Taste, doch glaube ich, das ist bekannt. Farbig gestalten kann man die Graphik entweder durch Einfügen eines entsprechenden Steuerzeichens in die PRINT-Anweisung, oder durch POKEn eines entsprechenden Wertes in die zur Bildschirmstelle korrespondierende Adresse des Color-RAMs.

Gehen wir davon aus, daß die Zeilen/Spalten von oben nach unten von 0 bis 24 und von links nach rechts von 0 bis 39 durchnumeriert sind, so läßt sich die Adresse des Color-RAMs wie folgt bestimmen:

Adresse = 55296 + 40 * Zeile + Spalte

Der Wert, der eingePOKEt werden kann, darf zwischen 0 und 255 liegen, allerdings bedeutet das nicht, daß es auch 256 verschiedene Farben gibt. Verantwortlich für die Farbgebung sind lediglich die vier niederwertigsten Bits, so daß wir aus 16 verschiedenen Farben auswählen können.

Mit welcher Zahl welche Farbe angesprochen wird, entnehmen Sie bitte dem Handbuch.

Der Vorteil einer solchen einfachen Blockgraphik besteht darin, daß sie einerseits einfach zu generieren, andererseits aber auch einfach zu verschieben ist (sie besteht ja lediglich aus 1000 Zeichen). Der Nachteil ist die Tatsache, daß diese Graphik nur ein sehr grobes Bild aufbauen kann.

3.3. Hochauflösende Graphik

Sie ist das Non Plus Ultra in punkto Bildschirmgestaltung. Wählt man die hochauflösende Grafik, so kann man zwischen 2 verschiedenen Modi wählen.

1. Normal-Modus

Er bietet eine Auflösung von 320 * 200 Punkten. Man kann 16 verschiedene Farben verwenden, allerdings nur mit gewissen Abstrichen. Hat man sich entschieden, daß ein Punkt z.B. rot ist, so sind alle übrigen Punkte innerhalb derselben 8 * 8 Matrix (ein Relikt der Blockgraphik) ebenfalls rot. Dies

hängt mit der Speicherorganisation zusammen. Mit dem Einschalten der Hires-Graphik (wie man diesen Modus auch noch nennt) verliert das eigentliche Color-RAM seine Bedeutung. An seine Stelle tritt das Video-RAM. Sie haben richtig gelesen. Der Bereich, der vormals für die Buchstaben auf dem Bildschirm verantwortlich war, liefert jetzt die Farbe. Jeweils ein Byte ist für die Farbinformation von 8 * 8 Punkten (oder Pixeln) verantwortlich. Die vier höherwertigen Bits geben dabei die Farbe der Punkte an, die anderen vier die Farbe des Hintergrundes. Die Information, welche Punkte gesetzt sind, entnimmt das Betriebssystem der sogenannten Bit-Map. Sie steht im RAM ab der Adresse 8192, allerdings nur, wenn die Graphik eingeschaltet ist. Im übrigen muß dieser Bereich dann geschützt werden. Das sind 8000 Bytes, wobei jedes Byte für jeweils 8 Punkte verantwortlich ist.

Die Ordnung der Bytes ist wie folgt: 8 Bytes stehen untereinander und bilden so die Matrix eines normalen Zeichens, das 9. Byte steht rechts neben dem 1. und so weiter, bis die Zeile gefüllt ist. Nach diesem Schema ist die gesamte Bit-Map auf dem Bildschirm wiederzufinden. Diese Tatsache macht es nicht ganz einfach, mit der Hires-Graphik zu arbeiten.

2. Multi-Color-Modus

Grundsätzlich gilt für diesen Modus ähnliches wie für die Hires-Graphik. Auch hier wird die Punktinformation der Bit-Map entnommen. Der Unterschied ist folgender: Diesmal bilden 2 Bits einen Punkt auf dem Bildschirm, die Auflösung verringert sich also auf 160 * 200 Punkte. Der Vorteil, der dadurch erkauft wird, ist die Möglichkeit, verschiedenfarbig zu arbeiten.

Mit dem Einschalten des Multi-Color-Modus gewinnt das alte Color-RAM wieder Bedeutung. Zusammen mit dem Video-RAM ist es für die Farbinformation verantwortlich. Die Auswahl, aus

welchem Speicherbereich die Farbinformation genommen wird, geschieht wie in der gleich folgenden Tabelle angegeben.

Jeweils 2 Bits ergeben einen Punkt auf dem Bildschirm. Mit 2 Bits lassen sich 4 verschiedene Zustände erreichen:

00: Hier wird die Farbe aus dem Hintergrundregister (53281) genommen.

01: Farbinformation aus den 4 höherwertigen Bits des Video-RAMs.

10: Farbinformation aus den 4 niederwertigen Bits des Video-RAMs.

11: Farbinformation aus dem Color-RAM

3.3.1. Einschalten der hochauflösenden Graphik

Kommen wir nun zu einer Sache, die Sie sicher schon brennend interessiert hat, um so mehr, als darüber im Handbuch des C-64 keine Silbe verloren wird.

Verantwortlich für alles, was auf dem Bildschirm erscheint, ist der VIC (Video Interface Chip). Er besitzt für die Steuerung seiner Aufgaben verschiedene Register, mit denen wir auch die hochauflösende Graphik einschalten können. Dieses geschieht-wie für den Commodore typisch-mittels eines POKE-Befehls.

Hier die Befehle, die zum Einschalten der Graphik führen:

POKE 43, 65: POKE 44, 63: POKE 16192, 0

Diese Befehle sorgen dafür, daß der Speicherbereich, den die Bit Map einnimmt, geschützt wird.

POKE 53265, 59

Hier wird der Graphikbildschirm eingeschaltet.

POKE 53272, PEEK (53272) OR 2^3

Legt die Lage der Bit Map fest. In diesem Fall liegt sie, wie bereits erwähnt, ab 8192. Die andere Alternative wäre, die Bit Map ab der Adresse 0 zu positionieren. Nur erklären Sie bitte dem Rechner, daß er von jetzt ab ohne Zero-Page leben muß.

Wenn Sie bis jetzt alles eingetippt haben, dürfte sich zur Zeit das totale Chaos auf Ihrem Bildschirm breit machen. Kein Wunder, denn die Bit Map muß erst zurückgesetzt werden. Das erledigt für uns folgende Schleife:

```
FOR I = 0 TO 7999

POKE 8192 + I, 0: NEXT I
```

Jetzt sieht die Sache schon anders aus. Nur noch ein paar farbige Klötze fallen störend ins Auge. Wir erinnern uns, das ehemalige Video-RAM ist für die Farbgebung der Hires-Graphik verantwortlich. Was wir sehen ist also nichts anderes, als die Bytes, die noch im Video-RAM stehen, jetzt allerdings eine andere Aufgabe erfüllen.

Mit folgender Schleife veranlassen wir auch diese, einen von uns gewünschten Wert anzunehmen.

```
FOR I = 0 TO 999

POKE 1024 + I, 16 * Punktfarbe + Hintergrundfarbe.

NEXT I
```

Es ist vollbracht. Die hochauflösende Graphik ist eingeschaltet.

Wollen wir allerdings den Multi-Color-Modus einschalten, so müssen wir noch ein bißchen mehr eintippen.

```
POKE 53270, 216
```

Setzt den Mult-Color-Modus.

```
FOR I = 0 TO 999

POKE 55296 + I,Farbe : NEXT I
```

Setzt die Farbe, die für die Bit-Kombination 11 gültig ist.

Fehlt nur noch die Information, um die Graphik wieder auszuschalten.

POKE 53265, 155: Graphik-Modus ausschalten

POKE 53270, 8 : Multi-Color-Modus ausschalten

POKE 53272, 21 : Großschrift

Das wäre das Wesentliche über die hochauflösende Graphik. Für Spiele hat sie trotz ihrer hervorragenden Auflösung nur geringe Bedeutung, da ihre Handhabung zu kompliziert ist, außerdem benötigt sie einen bedeutend größeren Speicherbereich.

Wenn jemand über ein Graphikunterstützendes Hilfsprogramm verfügt, so steht es ihm frei, damit Hintergründe zu zeichnen. Er wird sicherlich schöne Resultate erzielen. Allen anderen möchte ich ans Herz legen, sie für Spiele zu vergessen, wir werden eine Technik kennenlernen, die fast ebenso gute Resultate liefert, aber bedeutend benutzerfreundlicher ist.

Hochauflösende Graphik:

Vorteile: Gute Auflösung, damit verbunden gute Gestaltungsmöglichkeiten.

Nachteile: Großer Bedarf an Speicherplatz; komplizierte Programmierung. Nahezu ungeeignet für Tele-Spiele mit bewegtem Hintergrund (das Verschieben von 10 K RAM dauert auch in Maschinensprache seine Zeit).

3.4. Blockgraphik mit selbstdefinierten Zeichen

Ähnlich wie bei der hochauflösenden Graphik gibt es auch im Zeichenmodus verschiedene Betriebsarten. Hier sind es derer drei: der Normalmodus, der Extended-Color-Modus und der Multi-Color-Modus. Zum ersten ist nicht viel zu sagen. Er dürfte bekannt sein, ist er doch der normale Zustand, in dem der C-64 üblicherweise zu finden ist: Jedes Zeichen kann 2 Farben annehmen (Zeichenfarbe und Hintergrundfarbe), die Zeichenfarbe läßt sich für jedes der 1000 Zeichen auf dem Bildschirm einzeln wählen. Interessanter sind da schon die anderen Modi. Doch zuvor einige Informationen, wie so ein Buchstabe entsteht, der schließlich auf dem Bildschirm zu bewundern ist. Unser C-64 besitzt in dem sogenannten Character-Generator einen ROM-Bereich von 4 K Länge. In diesem Bereich ist nichts anderes abgespeichert, als die Form der Zeichen, wie sie auf dem Bildschirm zu sehen sind. Jeweils 8 Bytes ergeben die Form eines Zeichens. Der Character Generator besteht also aus nichts weiter als aus den Informationen für 4096 : 8 = 512 Zeichen. Um eigene Zeichen definieren zu können, müssen wir nichts anderes machen, als diesen ROM-Bereich ins RAM zu kopieren, und dem Betriebssytem mitteilen, wo er den Character-Generator nach der Verlegung findet.

Doch bevor wir dies tun, erst zu den anderen Betriebsarten des Normalmodus.

3.4.1. Extended-Color-Modus

Der Extended-Color-Modus weist viele Parallelen zum Normalmodus auf. Auch hier ergibt jedes Bit des Zeichenmusters einen Punkt der Farbe, die im Color-RAM zu finden ist, auf dem Bildschirm. Die nicht gesetzten Bits

können jedoch verschiedene Farben annehmen, und zwar je nachdem eine der Farbinformationen der Hintergrundregister 0 bis 3 (53281 - 53284). Welches dieser Register für die Farbgebung ausschlaggebend ist, hängt von dem Zeichencode ab. Interessant sind hierfür die beiden höchsten Bits. Sie dienen quasi als 2-Bit-Zahl als Zeiger auf die verschiedenen Hintergrundregister (Bitkombination 11 zeigt also auf das Register 3; 01 auf das Register 1). Damit wird auch schon ein Nachteil dieses Modus deutlich: Wenn diese beiden Bits als Zeiger für die Farbe dienen, können nur noch die restlichen 6 Bits als Zeiger für das entsprechende Zeichen verwendet werden. Es können also nur noch 2^6 = 64 verschiedene Zeichen angesprochen werden.

Einschalten des Extended-Color-Modus:

```
POKE 53265, PEEK (53265) OR 64
```

Das Ausschalten geschieht durch:

```
POKE 53265, PEEK (53265) AND 191
```

3.4.2. Multi-Color-Modus

Auch dieser Modus erlaubt mehrfarbige Zeichen. Es können bis zu 4 Farben je Zeichen verwendet werden, jedoch nur mit einer eingeschränkten Matrix der Zeichen. Jeweils 2 Bits ergeben die Information für einen Punkt auf dem Bildschirm. Verantwortlich dafür, ob ein Zeichen im Multi-Color-Modus ausgegeben wird, ist das Bit 3 im Color-RAM. Ist es an der entsprechenden Stelle des RAMs natürlich gesetzt, so erscheint das Zeichen mehrfarbig. Da dieses Bit als Zeiger auf den Multi-Color-Modus gebraucht wird, sind also nur noch 3 Bits für die Farbgebung da. Wir können jetzt nur noch die Farben 0 bis 7 ansprechen.

Die Farbe der Punkte richtet sich nach der Bit-Kombination, die den Punkt ausmacht.

Bit-Kombination : Farbe

00: Hintergrundfarbe (Hintergrundfarbregister 0)

01: Hintergrundfarbregister 1

10: Hintergrundfarbregister 2

11: Farbe aus dem entsprechenden Byte des Color-RAM.

Einschalten des Multi-Color-Modus:

POKE 53270,PEEK (53270) OR 16

Zusätzlich muß im Color-RAM noch das Bit 3 gesetzt sein.

FOR I = 0 TO 999

POKE 55296 + I, PEEK (55296 + I) OR 2^3

NEXT I

Das Ausschalten geschieht mit:

POKE 53270, PEEK (53270) AND 239

Es lohnt sich, mit dem Multi-Color-Modus ein wenig zu experimentieren, da er bei selbstdefinierten Zeichen hervorragende Ergebnisse liefert.

3.5. Zeichen selbst definieren

Die Form eines Zeichens richtet sich nach dem Character-Generator. Hier sind die Informationen für insgesamt 512 Zeichen abgespeichert (je 256 für den Klein- und Großschriftmodus). Um selbst Zeichen gestalten zu können, müssen wir also diese Information verändern. So ohne weiteres ist dies nicht möglich, befindet sich der Character-Generator doch im ROM und ist somit gegen POKEn resistent.

Wir müssen also den Character-Generator verlegen und dem Betriebssystem mitteilen, wo er in der neuen Situation seine Zeichen findet.

Bevor wir das Character-ROM verlegen, müssen wir es erst auslesen und an einer neuen Stelle im RAM abspeichern. Das Character-ROM liegt an einer Stelle im Adressbereich des C-64, die dreifach genutzt wird. Es wird vom Betriebssystem wechselweise mit dem I/O-Bereich angesprochen. Durch Löschen des Bits 2 in der Adresse 1 (siehe Teil 1) sprechen wir es an. Das Löschen dieses Bits ist aber insofern problematisch, weil dem Betriebssystem damit gleichzeitig der I/O-Bereich entzogen wird. Dies würde aber zum Absturz des Rechners führen, sobald dieser Bereich benötigt wird (und das wird er sogar recht häufig). Regelmäßig im Interrupt wird der I/O-Bereich eingeschaltet, um Tastaturabfrage und ähnliches mehr zu erreichen.

Wir müssen also folgendes beachten:

1. Wir müssen die Tastaturabfrage bzw. überhaupt den gesamten I/O Bereich vorübergehend "schließen".

2. Wir müssen den Character-Generator für uns lesbar machen.

3. Verlegen des Character-Generators.

4. Schützen des neuen Character-Bereiches.

5. Dem Betriebssystem mitteilen, wo er den neuen Character-Generator findet.

6. I/O-Bereich wieder zugänglich machen.

Das sieht nach einer Menge Arbeit aus, aber der Schein trügt. In Wirklichkeit ist die Sache schnell vergessen.

Auf den ersten Blick gibt es zwei Probleme: Die Sperrung des I/O-Bereiches und die Mitteilung an das Betriebssystem, wo der Character-Generator zu finden ist.

Indem wir den Interrupt ausschalten, verhindern wir, daß der I/O-Bereich angesprochen wird. Mit dieser zugegeben etwas brutalen Methode wird erreicht, daß das Betriebssystem lahmgelegt wird. Gleichzeitig machen wir aber auch die Tastaturabfrage unmöglich (sie ist ja eine Aufgabe des Betriebssystems), so daß das Kommando nicht im Direkt-Modus gegeben werden sollte. Damit wäre das erste Problem gelöst.

Die Nachricht an das Betriebssystem ist genauso einfach:

Wie über viele andere Dinge auch, so verfügt der C-64 auch hier über einen elektronischen Briefkasten. Es ist die Speicherzelle 53272. Hier geben die Bits 1 - 3 die Lage des Character-Generators wieder. Eine Bemerkung noch: Sind Hilfsprogramme oder ähnliches geladen, so ist es durchaus möglich, daß das folgende nicht funktioniert. Die Anleitung bezieht sich nur auf den Normalzustand des Rechners (es gibt noch andere Techniken, den Zeichengenerator zu verlagern, doch sind diese wesentlich komplizierter).

Kommen wir zurück auf das Byte 53272. Die nun folgende Tabelle verdeutlicht, bei welcher Bit-Kombination das Betriebssystem wo den Character-Generator sucht.

Bit 3 2 1 : Lage des Generators.

0 0 0 : Ab Adresse 0

0 0 1 : Ab Adresse 2048

0 1 0 : ROM (Großschrift)

0 1 1 : ROM (Kleinschrift)

1 0 0 : Ab Adresse 8192

1 0 1 : Ab Adresse 10240

1 1 0 : Ab Adresse 12288

1 1 1 : Ab Adresse 14336

Wir sehen, die Kombinationen 010 und 011 nehmen Sonderstellungen ein. Sie adressieren einen Bereich, in den unter normalen Umständen der Character-Generator ins RAM eingespiegelt wird. Dies sollte uns aber nicht weiter interessieren.

Die für uns bedeutsamste Kombination ist für uns die zweite, 001. Sie zeigt an den Basic-Anfang, adressiert also einen Bereich, der für uns leicht zu schützen ist.

Damit wäre der Weg beschrieben, machen wir uns ans Verlegen.

1. Schützen des neuen Bereiches

POKE 43, 1: POKE 44, 24: POKE 6144, 0: NEW: CLR

2. Verlegen des Character Generators

Hier muß ein kleines Basic-Programm her:

```
10 POKE 56334, PEEK (56334) AND 254: Interrupt ausschalten
20 POKE 1, PEEK (1) AND 251: ROM selektieren.
30 FOR I = 0 TO 4095
40 POKE 2048 + I, PEEK (53248 + I)
50 NEXT I
60 POKE 1, PEEK (1) OR 4: ROM ausschalten.
70 POKE 56334, PEEK (56334) OR 1: Interrupt einschalten
```

So, wir haben den Character-Generator ins RAM kopiert. Es fehlt also nur noch ein Befehl: Das Umschalten auf den neuen Bereich.

POKE 53272, (PEEK (53272) AND 241) OR 2

Und was passiert? Nichts! Enttäuscht?

Nicht nötig, wie sollte denn etwas passieren, wenn die Buchstaben nach wie vor aus demselben Bitmuster generiert werden.

Machen Sie einmal folgendes: Tippen Sie NEW ein, und löschen Sie den Bildschirm. Haben Sie es? Dann tippen Sie bitte folgendes Programm ein:

```
10 PRINT "CLR/HOME"
20 PRINT "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
30 POKE 198, 0: WAIT 198,1
40 FOR I = 8 TO 15
50 READ A: POKE 2048 + I, A
60 NEXT I
70 DATA 126,129,165,0,24,24,66,60
```

Wenn Sie fertig sind, starten Sie bitte das Programm und drücken dann auf eine Taste. Schön, nicht wahr?

Wenn Sie die alten As wiederhaben wollen, müssen Sie die Zeile 70 wie folgt ändern:

70 DATA 24,60,102,126,102,102,102,0

Sie können dasselbe mit der Tastenkombination RUN STOP/RESTORE erreichen, haben dann allerdings wieder auf das Character-ROM umgeschaltet.

Um gezielt Zeichen verändern zu können, brauchen Sie noch folgende Information: Das erste der 8 Bytes, die ein Zeichen ausmachen, finden Sie mit Hilfe folgender Formel:

Adresse = 2048 + 8 * ASC (Zeichen)

Die soeben erschlossene Graphik ist wie geschaffen, um sie in Telespielen einzusetzen, da sie

- durch PRINT Befehle installierbar ist,

- nur 2K für Bildschirm und Farbe beötigt,

- leicht zu handhaben ist.

Zum Abschluß dieses Kapitels möchte ich Ihnen empfehlen, ein wenig "herumzudefinieren", Sie werden sehen, es macht Spaß und, vor allem, es lohnt sich.

4. Sprite-Graphik

Wie kaum ein anderer Computer ist der C-64 durch die Möglichkeit der Sprite-Graphik zur Spieleprogrammierung besonders geeignet. In diesem Kapitel wollen wir einmal die Technik der Sprite-Programmierung aufdecken.

Obwohl in der Werbung stark hervorgehoben, wird die Programmierung der Sprites vom Handbuch, wie auch alles übrige, eher stiefmütterlich behandelt. Es wird zwar beschrieben, wie ein Sprite generiert wird, damit hört es aber auch schon auf. Um etwas Licht in die Angelegenheit zu bringen, hier eine Auflistung der Möglichkeiten, die durch die Sprite-Graphik geboten werden.

4.1. Sprites einschalten

Wie bekannt sein dürfte, kann der C-64 acht Sprites kontrollieren. Für diese Sprites sind ab der Adresse 53248 etliche Register reserviert, mit denen man sie bewegen, vergrößern, färben und kontrollieren kann. Dieses Unterkapitel wird sich, wie die folgenden auch, mit diesen Registern beschäftigen. Eine Auflistung finden Sie im Handbuch unter dem Anhang N.

Bevor wir jetzt mit den Sprites umgehen können, müssen wir sie erst einmal einschalten. Für diesen Zweck gibt es das Register 53269. Hier ist für jedes Sprite ein Bit vorgesehen. Ist es gesetzt, so ist das entsprechende Sprite eingeschaltet.

```
POKE 53269, PEEK (53269) OR 2^X
```

schaltet Sprite X ein (X ist ein Wert zwischen 0 und 7).

POKE 53269, PEEK (53269) AND (255 - 2^X)

schaltet Sprite X aus.

Zu jedem Sprite existiert zusätzlich ein Zeiger, der dem VIC (das ist der Baustein, der die Sprites generiert) mitteilt, wo die Information für die Spriteform zu finden ist. Es handelt sich um die Register 2040 bis 2047. Der hier abgelegte Wert ergibt mit 64 multipliziert die Adresse, ab der die 63 Bytes abgelegt sind, die das Sprite in seiner Form bestimmen. Für ein etwas aufwendigeres Programm kann es schnell vorkommen, daß man mehr als 3 Sprite-Blöcke benötigt (diese könnte man im Kasettenpuffer ablegen), so daß man sich frühzeitig Gedanken darüber machen sollte, wo man die Sprite-Informationen hinschreibt. Am besten ist es, wenn man den Basic-Start verschiebt und dadurch einen Speicherbereich schützt. Zu beachten ist, daß der höchste Block, der durch die Zeiger angesprochen werden kann, der Block 255 ist. Dies bedeutet, daß man die Sprite-Daten nicht an beliebiger Stelle ablegen kann, sondern nur bis zur Adresse 16320 (erstes Byte des Sprite-Blocks 255).

Eine kleine Information noch: Sollte sich trotz des Einschaltens eines Sprites in der Adresse 53269 und des Setzens eines entsprechenden Zeigers in einer der Adressen ab 2040 auf dem Bildschirm nichts "tun", so heißt das noch lange nicht, daß Ihr Gerät defekt ist, da das Sprite eventuell eine Position einnimmt, die außerhalb des Bildschirmfensters liegt. Mehr darüber aber im Kapitel Sprites bewegen.

4.2. Vergrößern von Sprites

Hat man erst einmal ein Sprite generiert, so bietet der C-64 viele Möglichkeiten, mit diesem zu arbeiten. Eine dieser

Möglichkeiten wäre z.B. das Verdoppeln der Sprite-Größe in X- und/oder Y-Richtung.

Hierfür dienen die Adressen 53271 (für die Y-Richtung) und 53277 (für die X-Richtung). Numeriert man die Sprites von 0 bis 7 durch, so läßt sich jedes Sprite durch

POKE 53277, PEEK (53277) OR 2^ Spritenummer

in X-Richtung; durch

POKE 53271, PEEK (53271) OR 2^ Spritenummer

in Y-Richtung vergrößern.

Die Sprites lassen sich durch:

POKE 53277, PEEK (53277) AND (255 - 2^ Spritenummer)

bzw.

POKE 53271, PEEK (53271) AND (255 - 2^ Spritenummer)

wieder verkleinern.

4.3. Farbgebung

Jedes Sprite kann eine eigene Farbe annehmen. Es gibt für jedes Sprite ein Register, in der die Farbe abgelegt wird.

POKE 53287 + X, Farbe

färbt das Sprite X mit der entsprechenden Farbe. Als Farbwert darf jeder beliebige Wert zwischen 0 und 15 gewählt werden.

4.4. Multi-Color-Sprites

Auch wenn Sie jetzt staunen, es ist tatsächlich möglich, mehrfarbige Sprites zu erzeugen. Jedes Sprite kann einzeln als Multi-Color-Sprite definiert werden, dabei verändert sich allerdings die Punktmatrix auf 12 * 21 Punkte je Sprite. Hier bilden also ähnlich wie im entsprechenden Modus des Zeichensatzes 2 Bits einen Punkt.

Jedes Multi-Color-Sprite besitzt eine eigene Grundfarbe, alle zusammen dieselbe zweite bzw. dritte Farbe. Für die Zweit- und Drittfarbe dürfen allerdings nur Werte von 0 bis 7 verwendet werden.

Auch hier sind wieder die Bitkombinationen ausschlaggebend dafür, welche Farbe der entsprechende Punkt erhält.

00 : Bildschirmhintergrund

01 : Multi-Color-Farbe 1 (Register 53285)

10 : Sprite-Farbregister

11 : Multi-Color-Farbe 2 (Register 53286)

Sicher interessiert Sie jetzt noch, wie ein normales Sprite zu einem Multi-Color-Sprite wird.

Das verantwortliche Register besitzt die Adresse 53276, durch

POKE 53276, PEEK (53276) OR 2^X

wird Sprite X zu einem Multi-Color-Sprite. Ausschalten läßt es sich

```
POKE 53276, PEEK (53276) AND (255 - 2^X)
```

4.5. Priorität

Wenn Sie ein Sprite auf dem Bildschirm positionieren, so wird der Eindruck erweckt, das Sprite bewege sich über den Hintergrund. Ein Buchstabe, der an derselben Position steht wie ein Sprite, wird also von diesem verdeckt. Dies ist in den meisten Fällen ganz praktisch. Es gibt aber auch Situationen, in denen das sehr stören kann. Stellen Sie sich vor, eine Situation in einem Ihrer Spiele erfordert es, daß ein Auto unter einer Brücke durchfährt. Es wäre also ganz nützlich, wenn man bestimmen könnte, ob ein Sprite "vor" oder "hinter" den Zeichen auf dem Bildschirm bewegt wird. Diese Möglichkeit gibt es. Die Regelung geschieht durch das Register 53275. Ähnlich wie bei dem Multi-Color-Register ist auch hier jeweils ein Bit für jedes Sprite reserviert. Wird ein Bit gesetzt, so verschwindet das entsprechende Sprite hinter den Zeichen auf dem Bildschirm.

```
POKE 53275, PEEK (53275) OR 2^X
```

läßt Sprite X hinter den Hintergrund tauchen.

```
POKE 53275, PEEK (53275) AND (255 - 2^X)
```

gibt dem Sprite die normale Priorität.

Ähnlich wie zwischen Sprite und Hintergrund gibt es auch zwischen den Sprites eine Rangfolge. Diese ist allerdings vorgegeben und läßt sich nicht von uns beeinflussen. Die existierende Reihenfolge ist folgende: Jedes Sprite mit einer niedrigen Spritenummer verdeckt alle Sprites, die über eine höhere Nummer verfügen.

9

```
10 REM ********* SPRITE-DEMO *********
15 REM ********* SPRITE DATAS *********
20 DATA1,128,0,3,192,0,7,224,0,15,240,0,
31,248,0,63,252,0,127,254,0,255,255,0
30 DATA255,255,0,127,254,0,63,252,0,31,2
48,0,15,240,0,7,224,0,3,192,0,1,128,0
40 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
50 FORI=0TO63
60 READA:POKE832+I,A
70 NEXTI
80 V=53248:POKE53281,5:PRINT"
 PRIORITAET"
90 FORI=0TO15STEP2
100 POKEV+I,42:POKEV+I+1,98
110 NEXT:POKEV+23,0:POKEV+29,0
120 FORI=0TO3:POKE2040+I,13:POKEV+39+I,I
122 NEXT
125 FORI=4TO7:POKE2040+I,13:POKEV+39+I,I
-4:NEXT
126 PRINT"
"
127 PRINT"                               "
130 POKEV+27,255:POKEV+21,255
140 FORI=0TO15STEP2
150 POKEV+I,PEEK(V+I)+2*I:FORR=0TO100:NE
XTR,I
160 FORI=0TO15STEP2:FORR=0TO150:IFR=80TH
ENPOKEV+27,PEEK(V+27)-2↑(I/2)
170 POKEV+I,PEEK(V+I)+1:NEXTR:FORR=0TO10
0:NEXTR,I
200 FORI=0TO1000:NEXT:POKEV+21,0
210 PRINT"          X,Y-EXPAND
  "
220 POKEV,40:POKEV+2,80:POKEV+4,120:POKE
V+6,160:POKEV+21,15:FORI=0TO500:NEXT
230 POKEV+23,2:FORI=0TO400:NEXT
240 POKEV+23,10:FORI=0TO400:NEXT
250 POKEV+29,4:FORI=0TO400:NEXT
260 POKEV+29,12:FORI=0TO400:NEXT
270 GOTO270
```

Wenn Sie die Möglichkeiten, die durch die Priorität gegeben werden, einmal demonstriert haben wollen, so tippen Sie obiges Programm ab und starten es. Es wird Ihnen verdeutlichen, was durch den Text ausgesagt wird.

4.6. Sprites bewegen

Es gibt für jedes Sprite 2 Register, in denen die Koordinaten für die Position abgespeichert sind. Es sind die Register 53248 bis 53263. Diese Register sind immer paarweise für die X- und Y-Koordinate zuständig (d.h. 53248 kontrolliert die X-Koordinate von Sprite 0, 53249 die Y-Koordinate zu Sprite 0, 53250 die X-Koordinate zu Sprite 1,usw.). Der Wert, der dem Registerpaar zu entnehmen ist, entspricht jeweils der linken oberen Ecke des entsprechenden Sprites. Will man ein Sprite gezielt auf dem Bildschirm plazieren, muß man dabei beachten, daß die Sprite-Koordinaten nicht mit den Bildschirm-Koordinaten übereinstimmen. Soll die linke obere Ecke eines Sprites in die linke obere Ecke des Bildschirms positioniert werden, so müssen in das Registerpaar die Werte 24 (für die X-Richtung) und 50 (für die Y-Richtung) eingePOKEt werden.

Auf diese Art, durch POKEn von Werten in die entsprechenden Register, lassen sich die Sprites einfach über den Bildschirm bewegen. Experimentiert man ein bißchen herum, wird man bald enttäuscht feststellen, daß unser Sprite den rechten Rand des Bildschirms nicht erreicht. Je nach Programm wird ein Illegal Quantity Error der Bewegung ein Ende bereiten oder das Sprite verschwindet einfach, um kurze Zeit später am linken Bildschirmrand wieder aufzutauchen.

Was ist der Grund für dieses lästige Verhalten?

Die Antwort: Die Register, die für die X-Koordinate der Sprites verantwortlich sind, wie auch alle anderen Register, sind 8-Bit-Register. Diese Register lassen aber nur Zahlen von 0 bis 255 zu. Da unser Bildschirm allerdings in X-Richtung über 320 Positionen verfügt, ist das Verhalten unserer Sprites irgendwie verständlich. Aber keine Angst, an so etwas haben die Konstrukteure des C-64 natürlich auch gedacht, und als Antwort das Register 53264 geschaffen. Hier gibt es sozusagen für alle Sprite-X-Register ein neuntes Bit. Wollen wir unser Sprite über die "Grenze" bewegen, so muß in der Situation, in der das Sprite die X-Position 256 einehmen soll, das dem Sprite entsprechende Bit im Register 53264 gesetzt werden, "gleichzeitig" das X-Register des Sprites auf Null gesetzt werden.

```
POKE 53264, PEEK (53264) OR 2^X
```

setzt das "neunte" Bit des X-Richtungs-Registers.

Will man wieder an den linken Rand, so muß dieses Bit wieder gelöscht werden. Dies geschieht wie üblich mit:

```
POKE 53264, PEEK (53264) AND (255 - 2^X)
```

So ausgerüstet ist der C-64 in der Lage Sprites über einen Bereich von 512 * 256 Positionen zu steuern. Dieser Bereich ist um einiges größer als das Bildschirmfenster, so daß die Sprites also seitlich hinter dem "Vorhang" verschwinden können.

4.7. Veränderung der Form

Bei Spielen kommt es oft vor, daß die Spielfigur ihre Form ändert. Ein Beispiel hierzu wäre eine Figur, die von links nach rechts über den Bildschirm läuft.

Es gibt drei Techniken, wie man so etwas realisieren kann, aber nur eine davon hat praktischen Wert.

1. Technik: Immer wenn eine andere Form vonnöten ist, lädt man andere Werte in den aktuellen Sprite-Block und verändert so die Form. Eine zugegeben primitive und aufwendige Technik.

2. Technik: Man ersetzt das alte Sprite durch ein neues mit denselben (bzw. fast denselben) Koordinaten. Auch diese Technik ist nicht gerade das Gelbe vom Ei.

3. Technik: Man ändert die Pointer, die auf die aktuellen Spritedaten zeigen (Register 2040 - 2047). So kann man auf einfache Art und Weise schnell die Form eines Sprites ändern. Macht man sich die Mühe und speichert viele einzelne Positionen eines Bewegungsablaufes ein, so erreicht man ohne großen Programmieraufwand eine flüssige Bewegung auf dem Bildschirm.

Sie werden sich vielleicht fragen, warum ich die beiden ersten Techniken aufgeführt habe? Lediglich zur Verdeutlichung, wie man durch gedankenlose Arbeit viel Speicherplatz vergeuden und gleichzeitig Programme unnötig kompliziert und langsam machen kann.

4.8. Kollisionen

Eine herausragende Eigenschaft von Sprites ist, daß man Zusammenstöße zwischen verschiedenen Sprites bzw. Sprites und Hintergrund leicht kontrollieren kann. Für diesen Zweck gibt es zwei weitere Register im VIC.

4.8.1. Sprite-Sprite-Kollisionen

Berühren sich zwei Sprites irgendwo auf dem Bildschirm und zwar so, daß sich gesetzte Bits der Sprites überlappen (das Überlappen von transparenten Stellen zählt also glücklicherweise nicht als Kollision), so werden im Register 53278 die den Spritenummern entsprechenden Bits gesetzt. Durch Auslesen des Registers kann also leicht festgestellt werden, ob und vor allem welche Sprites sich berührt haben, und die entsprechenden Schritte können in die Wege geleitet werden. Zu beachten ist allerdings, daß die Bits auch dann noch gesetzt bleiben, wenn sich die Sprites schon lange nicht mehr berühren. Es empfiehlt sich also, kurz vor der Abfrage das Register zu löschen, damit kein Zusammenstoß registriert wird, der vielleicht schon einmal beachtet wurde.

4.8.2. Sprite-Hintergrund-Kollisionen

In der gleichen Art gibt es auch ein Register, das Zusammenstöße zwischen Sprites und normalen Zeichen registriert. Auch hier muß eine echte Überlappung stattfinden. Transparente Bereiche reichen nicht für eine Kollision aus. Das Register hat diesmal die Adresse 53279.

Auch hier wird das entsprechende Bit des an der Kollision beteiligten Sprites gesetzt. An Hand der für die Koordinaten verantwortlichen Register läßt sich dann abschätzen, welches Zeichen der "Kontrahent" des Sprites war. Im übrigen gilt auch hier das über das Register 53278 gesagte; die Bits müssen nach der Abfrage zurückgesetzt werden, um Fehler zu vermeiden.

5. Sound Programmierung

Ähnlich wie die Graphik werden auch die Möglichkeiten der Tonerzeugung durch das Handbuch nur sehr unzureichend erklärt. Nun ist allerdings die Begleitmusik auch nicht so wichtig, als daß hier eine ausführliche Erläuterung angebracht wäre: Ich möchte mich mit dem Nötigsten begnügen.

5.1. Der SID

Der SID (= Sound Interface Device) ist unser kleiner elektronischer Schmied, der für die Erzeugung der Töne verantwortlich ist. Wie der VIC, den wir aus dem letzten Kapitel her kennen, verfügt er über eine Anzahl von Registern, mit denen er die gewünschten Aufgaben erfüllt.

Diese Register sind im wesentlichen verantwortlich für die Frequenz des Tones, die Wellenform und die Lautstärke. Der SID ist in der Lage, drei verschiedene Stimmen zu betreiben, besitzt also die wesentlichen Register in dreifacher Ausfertigung.

Kommen wir zur eigentlichen Programmierung:

5.2. Lautstärke

Sie ist zwar nicht das wichtigste Element, das Musik auszeichnet, bei der Sound-Programmierung sollte sie aber immer als erstes festgelegt werden. Das geschieht durch das Register 54296. Die Lautstärke läßt sich nur gemeinsam für alle drei Stimmen bestimmen. Die Lautstärke regeln hierbei

nur die Bits 0 bis 3. Man hat also die Wahl zwischen 16 Abstufungen: Von 0 (Stumm) bis 15 ("Volles Rohr").

5.3. Die Hüllkurve

Die Hüllkurve liefert dem SID die Information über den gewünschten Tonverlauf, dieser wird in vier Bereiche aufgegliedert: Anschlag, Abschwellen, Halten und Ausklingen.

Anschlag: Dieser Bereich umfaßt den Moment des Einschaltens bis zur Erreichung der maximalen Lautstärke, er wird durch die vier höherwertigen Bits folgender Register bestimmt, der Bereich liegt zwischen hart (0) und weich (15):

Stimme 1: Register 5

Stimme 2: Register 12

Stimme 3: Register 19

Abschwellen: Durch die übrigen Bits wird dieser Abschnitt der Hüllkurve definiert. Er reicht von dem Erreichen der Höchstlautstärke bis zum Erreichen der Dauerlautstärke. Analog zum Anschlag gibt es auch hier 16 Abstufungen von Hart bis Weich.

Halten: Hier gibt es nichts von Bedeutung anzumerken. Das Wort erklärt sich von selbst. Es kann wiederum ein Wert zwischen 0 und 15 (stumm bis laut) angenommen werden. Zu finden ist er in den höherwertigen Bits für:

Stimme 1: Register 6

Stimme 2: Register 13

Stimme 3: Register 20

Ausklingen: Mit dem Löschen des Start/Stop-Bits (wo dies zu finden ist, davon später), wird dieser Bereich angesprochen. Er regelt das Ausklingen des Tones. Auch hier liegen die möglichen Werte zwischen 0 (schnell) und 15 (langsam); sie sind abgespeichert in den restlichen Bits der Register 6, 13 und 20.

5.4. Das Tastverhältnis

Wird die Wellenform Rechteck bzw. Puls gewählt, so muß ein Verhältnis zwischen Impuls und Pause eingePOKEt werden. Dieses Verhältnis liegt im Bereich zwischen 0 und 4095. Hierfür gibt es ein Registerpaar, allerdings sind im High-Register nur die Bits 0 bis 3 von Bedeutung.

Stimme 1: Register 2 (Low) und 3 (High)

Stimme 2: Register 9 (Low) und 10(High)

Stimme 3: Register 10 (Low) und 17 (High)

5.5. Die Frequenz

Die Frequenz gibt die Tonhöhe an; das beste wird sein, man entnimmt die entsprechenden Werte dem Anhang P des Handbuches. Eingeschrieben werden die Werte in die Register:

Stimme 1: Register 0 (Low) und 1 (High)

Stimme 2: Register 7 (Low)und 8 (High)

Stimme 3: Register 14 (Low) und 15 (High)

5.6. Die Wellenform

Für die verschiedenen Stimmen gibt es je ein Register für die Wellenform und zwar:

Stimme 1: Register 4

Stimme 2: Register 11

Stimme 3: Register 18

Wie bereits schon angedeutet, hat man die Wahl zwischen verschiedenen "Instrumenten". Auch hier eine Übersicht:

Wert: Wellenform

129 : Rauschen

65 : Rechteck

33 : Sägezahn

17 : Dreieck

Werden diese Werte in die entsprechenden Register gePOKEt, so wird die Tonerzeugung gleichzeitig gestartet, das Bit 0 ist nämlich das oben bereits erwähnte Start/Stop-Bit. Wird es gelöscht (aber bitte nur dieses Bit), so wird die Hüllkurvenphase "Ausklingen" eingeleitet.

Wird in dieses Register der Wert 8 eingeschrieben, so wird im übrigen eine Initialisierung der entsprechenden Stimme durch den SID durchgeführt.

Zum Abschluß möchte ich dann noch das Programm "Pink Panther" vorstellen. Es wird die Ihnen sicherlich bekannte Melodie erzeugen.

Sollten Sie sich über die große Anzahl der DATAs wundern, so möchte ich dazu folgendes sagen: Jeder Ton benötigt genau vier DATAs, je eins für Low und High-Byte der Frequenz, die restlichen beiden für die Dauer des Tons und die Pause bis zum nächsten Ton.

```
10 DATA41,101,75,0,43,219,300,225,49,58,
75,0,52,39,300,225
20 DATA41,101,75,0,43,219,225,15,49,58,7
5,0,52,39,225,15,69,157,75,0,65,181
30 DATA225,15,43,219,75,15,52,39,225,15,
65,181,75,15,62,5,750,15,58,138,150,15
40 DATA52,39,150,15,43,219,150,15,39,18,
150,15,43,219,750,525
50 DATA41,101,75,0,43,219,300,225,49,58,
75,0,52,39,300,225
60 DATA41,101,75,0,43,219,225,15,49,58,7
5,0,52,39,225,15,69,157,75,0,65,181,225
70 DATA15,52,39,75,0,65,181,225,15,87,18
2,75,0,82,201,1500,225
80 DATA41,101,75,0,43,219,300,225,49,58,
75,15,52,39,300,225
90 DATA41,101,75,0,43,219,225,15,49,58,7
5,0,52,39,225,15,69,157,75,0
100 DATA65,181,225,15,43,219,75,0,52,39,
225,15,65,181,75,0,62,5,750,15
110 DATA58,138,150,15,52,39,150,15,43,21
9,150,15,39,18,150,15,43,219,750,900
120 DATA87,182,225,15,78,36,75,15,65,181
,225,15,58,138,75,15,52,39,225,15
130 DATA43,219,75,15,62,5,75,15,58,138,2
25,15
```

```
140 DATA62,5,75,15,58,138,225,15
150 DATA62,5,75,15,58,138,225,15
160 DATA62,5,75,15,58,138,225,15
170 DATA52,39,150,15,43,219,150,15,39,18
,150,15,43,219,150,15,43,218,900,600
180 DATA52,39,150,15,43,219,150,15,39,18
,150,15,43,219,150,15,43,219,150,15
190 DATA43,219,1200,600
192 DATA52,39,225,15,43,219,225,15,39,18
,225,15,43,219,225,15,43,219,225,15
194 DATA43,219,1200,3000
200 S=54272
210 POKES+24,15
220 POKES+5,136:POKES+6,248:POKES+4,8
230 FORI=0TO82:READA,B,C,D
240 POKES+4,17:POKES,B:POKES+1,A
250 FORR=0TOC:NEXTR:POKES+4,16
260 FORR=0TOD:NEXTR,I:RUN
```

6. Der Joystick

Mit die geeignetetste Art, Spielfiguren auf dem Bildschirm zu steuern, ist wohl der Einsatz eines Joysticks. Bei dem C-64 bestehen zwei Möglichkeiten, den Steuerknüppel an den Rechner anzuschließen: Die beiden mit Port 1 bzw. Port 2 gekennzeichneten Buchsen an der rechten Seite des Gehäuses. Steuern kann man dann, je nach Auslenkung des Joysticks und Auslegung des Programms, in alle vier Richtungen. Zusätzlich gibt es noch einen Knopf, den man z.B. als Feuerknopf verwenden kann.

Die Möglichkeit der Abfrage ist durch folgende Technik gegeben: Im Inneren des Joysticks befinden sich fünf Schalter, die je nach Preis des Joysticks aus Folienschaltern oder, in exklusiveren Modellen, aus Mikroschaltern bestehen. Vier dieser Schalter sind kreuzförmig um den Steuerhebel angeordnet und werden durch eine einfache Mechanik betätigt, sobald der Hebel in die entsprechende Richtung bewegt wird. Der fünfte Schalter ist mit dem Feuerknopf gekoppelt und überwacht so die Betätigung.

Im Computer wird der Zustand jedes einzelnen Schalters durch jeweils ein Bit einer Speicherzelle dargestellt, so daß die Stellung des Joysticks in einem Byte abgefragt werden kann. Folgende Tabelle gibt Aufschluß darüber, wie man den Inhalt des Bytes auswerten kann:

Joystick	Bit	7	6	5	4	3	2	1	0
Port 1		1	1	1	Knopf	Rechts	Links	Oben	Unten
Port 2		0	1	1	Knopf	Rechts	Links	Oben	Unten

Das der Stellung des Joysticks entsprechende Bit wird dabei gelöscht.

Ist der Joystick an Port 1 angeschlossen, so stehen die Informationen in der Speicherzelle 56321. Um diese Zelle auslesen zu können, muß in der Adresse 56320 der Wert 127 stehen. Bei der Verwendung von Port 1 sollte man beachten, daß bei der Funktion "Rechts" die CTRL-Taste simuliert wird, welche bekanntlich den Ablauf von Basic-Programmen bremst. Der Grund für dieses Verhalten ligt darin, daß der C-64 einen Joystick lediglich als einen Teil der Tastatur ansieht.
Um den Joystick an Port 2 auswerten zu können, muß folgendes gegeben sein: die Adresse 56322 muß den Wert 224 enthalten. Ausgelesen werden kann die Information durch PEEK(56320).

Will man sich einen Steuerknüppel kaufen, so sollte man darauf achten, daß der Joystick möglichst gut in der Hand liegt und nach Möglichkeit auf dem Tisch zu befestigen ist.
Im übrigen läßt sich jeder Atari-kompatible Joystick auch an den C-64 anschließen.

7. Das Hexadezimalsystem

Das Hexadezimalsystem ist ähnlich wie das üblicherweise gebräuchliche Dezimalsystem ein Zahlensystem, nur mit dem Unterschied, daß es nicht nur Ziffern von 0 bis 9 kennt, sondern noch ein paar mehr, nämlich Ziffern von 0 bis 15.

Man hat sich darauf geeinigt, für die 6 zusätzlichen Ziffern die Buchstaben von A bis F zu verwenden.

Hex-Zahlen werden im allgemeinen mit einem vorgesetzten String- bzw. Dollarzeichen als solche gekennzeichnet.

Sie werden sich sicher fragen, was will ich mit einem neuen Zahlensystem, das alte ist gut genug. Die Antwort ist folgende: Bei der Programmierung in Maschinensprache, und darum werden wir kaum herumkommen, bietet das Hexsystem einige nicht zu verachtende Vorteile.

Wir haben gehört, daß unter jeder Adresse ein Datum mit einem Wert zwischen 0 und 255 abgespeichert sein kann. Geben wir dieses Datum mit einer Dezimalzahl an, so kommen Zahlen mit 1, 2 oder 3 Stellen in Frage. Bei der Verwendung des Hex-Systems hat die Zahl immer 2 Stellen (es gibt keine Hexzahlen mit ungerader Stellenzahl; aus 11=$A wird immer $0A). Dies vereinfacht die Bearbeitung von Maschinensprache-Programmen sehr. Ein anderer Vorteil ist folgender: Ein Byte besteht bekanntlich aus 8 Bits. Teilt man ein Byte in die 4 höherwertigen bzw. 4 niederwertigen Bits auf (man spricht dabei auch von "Nibbels"), so läßt sich aus jedem einzelnen Nibbel eine Hexziffer ableiten, die zusammen die dem Wert des Bytes entsprechende Hexzahl ergeben. (Versuchen Sie das mal im Dezimalsystem...)

Man teilt die Zahl durch 16, notiert sich den Rest, und teilt das Ergebnis solange weiter durch 16, bis das Ergebnis gleich Null ist.

Dazu zwei Beispiele:

```
45523:16 = 2845, Rest  3 entspricht $3

 2845:16 =  177, Rest 13 entspricht $D

  177:16 =   11, Rest  1 entspricht $1

   11:16 =    0, Rest 11 entspricht $B
```

Aus dieser Rechnung folgt, daß 45523 im Hexsystem $B1D3 entspricht.

```
51017:16 = 3188, Rest  9 entspricht $9

 3188:16 =  199, Rest  4 entspricht $4

  199:16 =   12, Rest  7 entspricht $7

   12:16 =    0, Rest 12 entspricht $C
```

Daraus folgt: Dezimal 51017 entspricht Hex $C749.

So, ich glaube diese Umrechnung dürfte nun klappen.

Kommen wir zum umgekehrten Weg: Zwei Beispiele für Umwandlung von Hex-Zahlen in Dezimal-Zahlen.

```
$DC37 = $D (=13)*4096 (=16^3)

+       $C (=12)* 256 (=16^2)

+       $3 (= 3)*  16 (=16^1)

+       $7 (= 7)*   1 (=16^0)
```

```
=               56375


$FE4A = $F (=15)*4096 (16^3)

+       $E (=14)* 256 (16^2)

+       $4 (= 4)*  16 (16^1)

+       $A (=10)*   1 (16^0)

=               65098
```

8. Das Binärsystem

Oh Gott, noch ein Zahlemsystem höre ich jetzt viele stöhnen. Richtig, aber auch das hat seinen guten Grund. Um effektiv programmieren zu können, brauchen wir ein Zählsystem, mit dem wir ein Byte bitweise betrachten können.

Das Binärsystem kennt nur zwei Ziffern: die 0 und die 1. Diese Tatsache bedingt eine unangenehme Nebenerscheinung des Binärsystems (oder Dualsystem, wie es auch noch genannt wird); selbst aus kleinen Zahlen werden riesige Zahlenkolosse, wenn man sie im Binärsystem aufschreibt.

Kommen wir jetzt zur Umwandlung. Sie erfolgt auf dem gleichen Weg wie die Umwandlung zum Hex-System, nur daß man diesmal nicht durch 16, sondern lediglich durch 2 teilt. Auch hier zwei Beispiele:

```
168:2 =  84, Rest 0

 84:2 =  42, Rest 0
```

```
 42:2 =  21, Rest 0

 21:2 =  10, Rest 1

 10:2 =   5, Rest 0

  5:2 =   2, Rest 1

  2:2 =   1, Rest 0

  1:2 =   0, Rest 1
```

Wir stellen also fest, 168 dezimal entspricht 10101000 binär.

```
237:2 = 118, Rest 1

118:2 =  59, Rest 0

 59:2 =  29, Rest 1

 29:2 =  14, Rest 1

 14:2 =   7, Rest 0

  7:2 =   3, Rest 1

  3:2 =   1, Rest 1

  1:2 =   0, Rest 1
```

Daraus folgt: 237 entspricht binär 11101101.

Gehen wir jetzt den umgekehrten Weg:

```
11001011 = 1 * 128 (=2^7)
```

```
+           1 *  64 (=2^6)

+           0 *  32 (=2^5)

+           0 *  16 (=2^4)

+           1 *   8 (=2^3)

+           0 *   4 (=2^2)

+           1 *   2 (=2^1)

+           1 *   1 (=2^0)

=               203


11100001 = 1 * 128 (=2^7)

+           1 *  64 (=2^6)

+           1 *  32 (=2^5)

+           0 *  16 (=2^4)

+           0 *   8 (=2^3)

+           0 *   4 (=2^2)

+           0 *   2 (=2^1)

+           1 *   1 (=2^0)

=               225
```

9. Binäre Arithmetik

Nachdem wir uns mit zwei neuen Zahlensystemen herumgeschlagen haben, wollen wir uns nun ein bißchen mit der Binär-Arithmetik beschäftigen.

Es gibt vier Verknüpfungsarten, die hier erläutert werden sollen: AND, OR, NOT und XOR (EOR). Wir werden jede dieser Verknüpfungen einmal auf ihre Wirkung hin untersuchen.

9.1. AND

Die AND- oder auch UND-Verknüpfung ist dann erfüllt, wenn beide Statements erfüllt sind. Sie kennen das vielleicht aus dem Basic: IF A=3 AND B=5 THEN 100

Bei der binären Verknüpfung sieht das ähnlich aus: Die beiden Zahlen werden bitweise verknüpft und das Ergebnis festgehalten.

Nehmen wir an, A entspricht der Binärzahl 11100110, B der Binärzahl 11011001, dann ist das Ergebnis für z.B. C=A AND B gleich:

```
A= 11100110 (=230)

B= 11011001 (=217)

...........

C= 11000000, oder Dezimal: 192
```

Ein weiteres Beispiel:

```
A= 11011111 (=223)

B= 11111001 (=249)

----------

C= 11011001 (=217)
```

9.2. OR

Die OR-Verknüpfung liefert ein wahres Ergebnis, wenn eines der beiden Statements richtig ist.

Auch hier zwei Beispiele:

```
A= 01110011 (=115)

B= 11000111 (=199)

---------

C= 11110111 (=247)


A= 10000000 (=128)

B= 00111111 (= 63)

---------

C= 10111111 (=191)
```

Wir haben jetzt zwei Verknüpfungen kennengelernt. Nun stellt sich die Frage, wofür wir sie brauchen. Die Antwort ist folgende: Bei bestimmten Bytes, vor allem in der Zero-Page, muß man sehr vorsichtig sein, will man diese verändern. Meist haben die einzeln Bytes völlig verschiedene Aufgaben und oft wirkt es sich katastrophal auf ein eventuell vorhandenes Programm aus, falls man ein falsches Bit setzt bzw. löscht. Mit den Verknüpfungen AND und OR sind dem Programmierer nun Instrumente an die Hand gegeben worden, mit denen ein Verändern dieser "gefährlichen" Bytes einfach und sicher vonstatten geht.

Einige Beispiele:

Um das letzte Bit einer Adresse zu löschen, verwendet man folgenden Befehl:

POKE A,PEEK(A) AND 254

Wir sehen, nur das letzte Bit (das Bit 0, es entspricht im gesetzten Zustand dem Wert von 2^0) wird verändert. Es wird auf jeden Fall gelöscht. Theoretisch könnte man auch vor dem Ablauf des Programms den entsprechenden Wert, den die Speicherzelle annehmen muß, ausrechnen und direkt in das Programm eingeben. Problematisch wird es nur, wenn die Speicherzelle "von sich aus", d.h. durch das Betriebssystem geändert wird. Dieser Fall kann verheerende Folgen sowohl für das Programm als auch für den Programmierer haben, darum, sicher ist sicher: Bitweise Veränderung von Bytes nur durch AND (bzw. OR).

Der Befehl OR wird verwendet, wenn einzelne Bits gesetzt werden solle ohne das gesamte Byte darüber hinaus zu verändern.

Der Befehl POKE A, PEEK(A) OR 1 setzt lediglich das Bit 0.

9.3. NOT

Die Verknüpfung NOT liefert genau dann ein positves Ergebnis (eine 1), wenn eines der beiden Statements wahr, das andere jedoch falsch ist. Diese Verknüpfung hat jedoch für uns keine große Bedeutung.

Hier wiederum 2 Beispiele: Not A bewirkt folgendes:

```
A= 11001100 (=204)

---------

B= 00110011 (= 51)


A= 10011101 (=157)

---------

B= 01100010 (= 98)
```

Wir sehen also, das ursprüngliche Byte wird einfach invertiert.

9.4. XOR (EOR)

Der Befehl XOR (EOR) ist im Basic V2.0 gar nicht vorhanden, trotzdem wird er uns in diesem Buch begegnen, da ein Maschinensprache-Befehl des 6510 diese Verknüpfung ausführt. Hier ist das Ergebnis genau dann wahr, wenn genau eines der beiden Statements wahr ist.

Wiederum zwei Beispiele:

```
A= 10101010 (=170)

B= 11111111 (=255)

----------

C= 01010101 (= 85)


A= 11000000 (=192)

B= 00111111 (= 63)

----------

C= 11111111 (=255)
```

10. Basic Kontra Maschinensprache

Selbst einfache, in BASIC geschriebene Spiele (z.B. Pelota in Kapitel 14) sind in Punkto Speed an der unteren Grenze des Erträglichen. Man stelle sich einmal vor, es gäbe zwei Schläger, wie zum Beispiel bei dem Spiel Tennis. Ergebnis: Das Spiel wäre absolut langweilig, weil zu langsam. Schon so, wie es aufgelistet ist, kann es kaum Interesse erwecken. Was wäre wohl, wenn man nun versuchen würde, ein Spiel wie Phoenix oder ein anderes "Weltraumspiel" in Basic zu programmieren. Mit jedem bewegten Motiv, das hinzu kommt, verlangsamt sich das Programm zusehends. Hier liegt auch das Problem, das viele, die eigene Spiele programmieren wollen, zur Aufgabe nötigt. Wir können also vorerst nur feststellen, daß sich Basic aufgrund seiner relativ geringen Geschwindigkeit nur bedingt für unsere Zwecke eignet. Verwendbar ist es für Programme, die mit wenig Aufwand auskommen. Wir werden solche Programme noch kennenlernen.
Es gibt aber auch noch eine andere, wenn auch ungleich kompliziertere Art, Programme zu erstellen: Die Programmierung in Maschinensprache.

10.1. Was ist "Maschinensprache"?

Das Herzstück eines jeden Computers ist die CPU, die Central Processing Unit, oder zu deutsch: der Microprozessor. Er verwaltet, steuert, rechnet, kurz er macht alles, was so ein Computer machen kann, und das mit einer irrsinnigen Geschwindigkeit. Könnte man bei einem Computer von Pulsschlag reden, so würde man feststellen, daß der Prozessor im C-64 mit ca. 980.000 Schlägen pro Sekunde arbeitet. Durch diese hohe Geschwindigkeit ist es möglich, daß ein Rechner so leistungsfähig ist. Im Grunde genommen ist nämlich so ein Prozessor "strohdoof". Er ist zwar in der Lage, eine große Zahl von Befehlen in einer kurzen Zeit abzuarbeiten, allerdings, ein solcher Befehl hat nicht viel mit einem Basic-Wort wie z.B. PRINT zu tun. Im Gegenteil: Es müssen viele Maschinensprachebefehle, denn nur solche versteht eine CPU, bearbeitet werden, bevor ein Basic-Befehl ausgeführt wurde. Daraus wird klar, warum Basic um so vieles langsamer ist als die Maschinensprache. Zur Verdeutlichung ein kleines Beispiel:

Tippen Sie mal folgendes kurze Programm ein:

```
10 For I=1024 To 2023
20 Poke I,1: Poke I+54272,0
30 Next I
```

Dieses Programm wird den Bildschirm mit schwarzen As füllen. Starten Sie bitte einmal das Programm, und stoppen Sie die Zeit, die benötigt wird, den Bildschirm zu füllen. Haben Sie es? Dann tippen Sie bitte folgendes ein:

```
10 Data 162,1,160,0,169,3,206,13,192
20 Data 206,16,192,142,232,7,140,232,219
30 Data 204,13,192,208,239,206,14,192,206
40 Data 17,192,205,14,192,208,228,96
50 For I=49152 To 49186:Read X:Poke I,X
60 Next I
```

Starten Sie bitte dieses Programm mit RUN. Zunächst wird gar nichts passieren. Tippen Sie nun SYS 49152 und halten sie die Stoppuhr bereit. Erstaunlich, nicht wahr? Haben Sie so etwas erwartet? Das war dasselbe Programm, in Maschinensprache geschrieben. Wenn Sie Ihren Augen noch nicht trauen, dann tippen Sie noch einmal RUN und dann wieder SYS 49152 ein. Das ist eine Geschwindigkeit, mit der man eher etwas anfangen kann, oder?
"Zerpflücken" wir das Programm einmal. Das eigentliche Programm sind die Zahlen hinter den DATA-Statements. Der Rest sorgt nur dafür, daß das Programm richtig eingelesen wird. In Assembler (darunter versteht man Abkürzungen, die für die einzelnen Befehle stehen) sieht das Programm wie folgt aus:

```
$C000 LDX#$ 01
$C002 LDY#$ 00
$C004 LDA#$ 03
$C006 DEC$C00D
$C009 DEC$C011
$C00C STX$07E8
$C010 STY$DBE8
$C013 CPY$C00D
$C016 BNE$C006
$C018 DEC$C00E
$C01B DEC$C012
$C01E CMP$C00E
$C020 BNE$C006
$C022 RTS
```

Diese Liste gibt das Programm in einer allgemein üblichen Liste von Abkürzungen wieder. So bedeutet z.B. "LDA" soviel wie load accumulator. Dies wurde eingeführt, um dem Menschen den Umgang mit der Maschine so einfach wie möglich zu gestalten. Man nennt dies Assembler oder eine Abkürzung für sich Opcode. Einen solchen Opcode gibt es für jeden Befehl. Sie ermöglichen ein einfacheres Hantieren mit den Befehlen, als z.B. Zahlen als Kennung. Opcodes sind nur eine Hilfe. Der Rechner kann nur mit den Zahlen arbeiten.

Das sieht unheimlich toll aus, nicht wahr? Ich glaube allerdings, daß ein Großteil der Leserschaft mit den obigen Zeilen nicht sehr viel anfangen kann. Übersetzen wir das Programm mal ins Deutsche:

$C000 Lade den Wert 1 ins X-Register
$C002 Lade den Wert 0 ins Y-Register
$C004 Lade den Wert 3 in den Akkumulator (auch ein Register)
$C006 Verkleinere den Wert in der Adresse C00D um Eins
$C009 Verkleinere den Wert in der Adresse C011 um Eins
$C00C Speichere den Inhalt des X-Registers in 07E8 ab
$C010 Speichere den Inhalt des Y-Registers in DBE8 ab
$C013 Vergleiche den Inhalt von C006 mit dem Y-Register
$C016 War der Wert gleich? Dann springe nach C006, sonst mache weiter
$C018 Verkleinere den Wert in der Adresse C00E um Eins
$C01A Verkleinere den Wert in der Adresse C012 um Eins
$C01D Vergleiche den Inhalt von C00E mit dem Akkumulator
$C020 War der Wert gleich? Dann springe nach C006, sonst mache weiter
$C022 Ende

Eine Erklärung noch: C000 bzw. C002 oder C00E sind Adressen, die im Hexadezimalsystem angegeben sind. Dieses Zählsystem ist bei Maschinensprache allgemein üblich, weil hier immer 2 Ziffern den Wert eines Bytes angeben. Verwendet man das Dezimalsystem so ist das nicht gegeben, da ein Byte einen Wert zwischen 0 und 255 verkörpern kann. Wie funktioniert das Hexadezimalsystem? Es gibt hier, ähnlich wie sonst auch, die Ziffern 0 bis 9, darüber hinaus aber auch noch die Buchstaben A bis F. Nun bedeuten die Buchstaben Zahlen von 11 (A) bis 15 (F). Hex-Zahlen sind durch ein vorgestelltes Dollarzeichen ($) als solche gekennzeichnet. Will man eine Hex-Zahl in eine "normale" Zahl umrechnen, so geht man wie folgt vor: $ABCD wird mit A*16^3+B*16^2+C*16+D in eine Dezimalzahl umgewandelt. Hierbei sind ABCD beliebige Hex-Ziffern.

Mit der Maschinensprache haben wir eine Möglichkeit, Routinen, die aus Geschwindigkeitsgründen in Basic nicht realisierbar sind, durchzuführen. Im folgenden wird eine kurze Einführung in die Maschinensprache gegeben.

11. Maschinensprache-Einführung

In den folgenden Abschnitten werden Sie sehr häufig Programme in Maschinensprache finden, die Ihnen zunächst vielleicht als Buch mit 7 Siegeln erscheinen. Das soll Sie nicht daran hindern, diese Programme fleißig anzuwenden. Wenn Sie aber eigene Spiele schreiben wollen, so kommen Sie um die Maschinensprache nicht herum (es sei denn, Sie möchten sich beim Spielen langweilen), denn BASIC ist einfach zu langsam. In diesem Fall sollten Sie sich eine spezielle Einführung für diese Programmierungstechnik zulegen, z.B. DATA-BECKER Maschinensprachebuch zum Commodore 64.

Damit Sie zumindest wissen, was Sie dann erwartet, folgt jetzt eine Art Mini-Lehrgang zur Maschinensprache. Ich hoffe, daß er Sie auf den Geschmack bringt.

11.1. Wie funktioniert eigentlich ein Computer?

Keine andere Programmiersprache hängt mit der Hardware so eng zusammen wie die Maschinensprache. Aber gerade weil sie vom Prozessor direkt verarbeitet werden kann (ohne Interpreter oder Compiler), ermöglicht sie auch so wahnsinnig hohe Geschwindigkeiten. Bevor wir uns also mit der eigentlichen Sprache befassen, sollten zumindest die Grundprinzipien der Funktionsweise eines Rechners bekannt sein. Dabei werden auch gleich ein paar (bisher vielleicht unbekannte) Begriffe geklärt.

Beginnen wir mit der Hardware. Die "Zentrale" Ihres 64ers ist der Mikroprozessor 6510. Er ist das einzige Bauteil, das wirklich Daten verarbeiten kann, alle anderen Chips dienen entweder dem Transport oder der Speicherung von Bytes oder wandeln diese in Töne und Bilder um.

Der 6510 kann maximal 64 Kilobytes Speicherbereich adressieren. Im C-64 sind aber allein schon 64 K RAM und dazu noch 20 Kilobytes ROM enthalten. Daher muß der Prozessor zwischen beiden Bereichen umschalten können (das nennt man übrigens Bankswitching). Welcher Bereich gerade eingeschaltet ist, wird vom Inhalt der Speicherzelle 1 gesteuert. Wer sich dafür näher interessiert, sollte sich mit einem der DATA BECKER Bücher (z.B. Peeks & Pokes, 64-intern etc.) befassen, in denen diese Technik beschrieben ist.

Für die Verbindung mit der Außenwelt sorgen diverse andere Bausteine. Die Aufgaben von VIC und SID dürften Ihnen bereits bekannt sein (siehe Commodore-Handbuch). Weiter gibt es noch zwei CIAs (Complex Interface Adapter). Das sind sogenannte Schnittstellenbausteine, an denen unter anderem die Tastatur, der USER-PORT und die Joysticks angeschlossen sind. Will der Prozessor ein Byte zu einer Schnittstelle schicken, so liefert er es einfach beim CIA ab und sagt ihm, was zu tun ist. Den Rest erledigt der Baustein alleine.

Wie läuft nun ein einzelner Maschinenbefehl intern ab? Zunächst muß der Befehlscode in den Prozessor geholt werden. Letzterer weiß, wo der Code gespeichert ist, und gibt deshalb die Adresse an den Speicherbaustein weiter. Zu diesem Zweck gibt es 16 Adressleitungen, die zusammen auch Adressbus genannt werden. Der Speicherbaustein "sucht" das richtige Byte heraus und gibt es über 8 Datenleitungen (Datenbus) an den Prozessor zurück. Der wiederum dekodiert den Befehl und führt ihn aus (z.B. Addition). Dabei kann es notwendig sein, Datenbytes aus dem Speicher zu holen. Das funktioniert genau wie beim Befehlscode.

11.2. Abschied von den Variablen

Eines der Grundprinzipien der Maschinensprache ist es, daß fast alle Datenmanipulationen im Prozessor ablaufen. Deshalb müssen zu verknüpfende Bytes erst in den 6510 geladen werden, bevor die eigentliche Verarbeitung (z.B. eine Addition) stattfinden kann.

Außerdem lassen sich nicht einfach Zahlen eingeben und in Variablen speichern. Die Maschinensprache kennt nur Speicherzellen und interne Register. Für die Unterscheidung von Programmbytes und Daten muß der Programmierer selbst sorgen. Um das ganze noch weiter zu verkomplizieren, können immer nur 8 Bits gleichzeitig bearbeitet werden. Größere Zahlen müssen in einzelne Bytes zerlegt und dann schrittweise bearbeitet werden.

Wenn Sie bisher der Meinung waren, daß man in Maschinensprache zumindest einzelne Bytes wie bei einem einfachen Taschenrechner multiplizieren und dividieren kann, so sind Sie leider auf dem Holzweg. Dafür sind eigene Programme nötig, die allerdings einfacher zu schreiben sind, als es der Außenstehende vermutet. Wollen Sie z.B. 3*4 berechnen, so können Sie dreimal die 4 addieren, denn 4+4+4=12. Die Addition beherrscht der 6510 hervorragend, ebenso die Subtraktion, womit auch Divisionen möglich werden. Wollen Sie 15 durch 3 dividieren, so zählen Sie einfach, wie oft 3 von 15 abgezogen werden kann, bis das Ergebnis 0 wird (nämlich 5 mal).

Im Prozessor stehen dem Programmierer 3 Register zur Verfügung. Der Akkumulator (oder kurz Akku) stellt das Hauptrechenregister dar. Alle Additionen, Subtraktionen und sonstigen Verknüpfungen haben hier ihren Ausgangspunkt und speichern ihr Ergebnis auch wieder im Akku ab.

Dabei dürfen die eineiigen Zwillinge X und Y nicht vergessen werden, die auch Indexregister genannt werden. Sie werden vor allem für die Arbeit mit Bytetabellen benutzt, weil es spezielle Befehle gibt, die den Inhalt von X bzw. Y zur Adresse des Tabellenanfangs addieren und so auf das x-te Byte zugreifen können.

Außerdem sind noch verschiedene Flags vorhanden, die den Zustand von Prozessorregistern bzw. Teile des letzten Ergebnisses signalisieren. Als Beispiele seien das Carry- und das Zero-Flag angeführt. Carry gibt an, ob bei der byteweisen Verarbeitung einer Zahl ein Übertrag entstanden ist, der dann ggf. im nächsten Schritt mitaddiert werden kann. Das Zero-Flag zeigt an, ob das letzte Ergebnis 0 war. So lassen sich sehr einfach Vergleiche programmieren. Wenn das Ergebnis aus A-B=0 ist, so bedeutet dies, daß beide Zahlen gleich waren.

11.3. Liste der Maschinensprachebefehle

Um Ihnen zu verdeutlichen, was in den Maschinenspracheroutinen dieses Buches geschieht, gibt es hier eine Erläuterung der im Buch verwendeten Befehle.

Maschinensprachebefehle lassen sich in 5 Gruppen unterteilen:

1. Befehle zur Bytemanipulation

2. Befehle zur Bitmanipulation

3. Vergleichsbefehle

4. Verzweigungsbefehle

5. Befehle zur Beeinflussung des Prozessors

1. Gruppe: Befehle zur Bytemanipulation

Diese Befehle dienen dazu, unsere Register mit Daten zu beschicken oder Daten, die in diesen Registern stehen, im RAM abzulegen.

LOAD-Befehle

Die LOAD-Befehle sind in gewisser Weise mit dem PEEK verwandt, auch sie lesen Adressen aus und geben das entsprechende Datum an ein Register weiter.

lda $nn:

Holt das Datum unter der Adresse $nn (16-Bit-Adresse) und legt es in den Akku ab.
In Basic sieht das dann so aus:
A = PEEK ($nn)

lda $n

bewirkt das gleiche wie der erste Befehl, mit dem Unterschied, daß die Adresse lediglich 8 Bit lang ist, man kann also lediglich die Adressen $00-$ff ansprechen. Der Vorteil liegt darin, daß der Befehl schneller arbeitet als jener mit der 16-Bit-Adresse.

lda #$n
Lädt den Wert $n in den Akku.
(LET A = $n)

lda $nn,x

Das Datum der Adresse ($nn + x-Register) wird in den Akku geladen. Es wird also zu der angegebenen Adresse noch der Inhalt des x-Registers addiert, und dann der Wert eingeladen.
(A = PEEK ($nn + X))

lda ($n),y

Dieser Befehl ist schon ein wenig aufwendiger zu erklären. Die Adresse $n ist das erste Byte eines 2-Byte-Pointers, der auf die Adresse zeigt, die schließlich eingeladen wird. Vorher wird allerdings zu der Adresse noch der Inhalt des y-Registers addiert und so die endgültige Adresse gefunden.
(A = PEEK (PEEK ($n) + 256 * PEEK ($n+1) + Y)

ldx $nn

Holt das Datum unter der Adresse $nn in das x-Register.
(X = PEEK ($nn))

ldx #$n

Lädt den Wert $n in das x-Register.
(LET X = $n)

ldy $nn

Holt das Datum unter der Adresse $nn in das y-Register.
(Y = PEEK ($nn))

ldy #$n

Lädt den Wert $n in das y-Register
(LET Y = $n)

Das waren die im Buch verwendeten LOAD-Befehle, der Prozessor verfügt noch über einige mehr. Eines haben allerdings alle LOAD-Befehle gemeinsam und das sollte man beachten: Sie beeinflussen die bereits erwähnten Status Flags (Negativ- und Zero-Flag werden je nach Datum gesetzt).

Die STORE-Befehle

Wenn man sagt, daß die LOAD-Befehle mit dem PEEK des Basic verwandt sind, dann ist es zweifellos naheliegend, die STORE-Befehle mit dem POKE zu vergleichen. Die in einem Register befindlichen Werte werden an einer Adresse im RAM abgelegt.

Der Syntax der STORE-Befehle ist, mit einer Ausnahme (ein Gegenstück zu lda #$n gibt es nicht), mit den LOAD Befehlen verwandt.

```
sta $nn        (POKE $nn, A)

sta $n         (POKE $n, A (8 Bit Adresse))

sta $nn,x      (POKE ($nn + x), A)

sta ($n),y     (POKE (PEEK ($n) + 256 * PEEK ($n+1) + Y), A)

stx $nn        (POKE $nn, X)

stx $n         (POKE $n, X)

sty $nn        (POKE $nn, Y)
```

```
stx $n       (POKE $n, Y)
```

Genau wie bei LOAD-Befehlen wird auch bei den STORE-Befehlen die Negativ- bzw. Zero-Flag verändert.

Befehle zum In- bzw. Dekrementieren

Es ist durchaus möglich, daß Sie mit der Überschrift nichts anfangen können. Was versteht man also unter dem In- bzw. Dekrementieren von Werten.

Inkremetierung:

inc $nn

Der Befehl bewirkt, daß der Inhalt der Adresse $nn um den Wert Eins erhöht wird.
Wird dabei der Wert 255 überschritten, so wird der Wert Null eingeladen und das Zero-Flag gesetzt. Zusätzlich kann durch Inkrementieren das Negativ-Flag gesetzt werden.
In Basic sähe der obige Befehl wie folgt aus:
POKE $nn, PEEK ($nn) + 1

inx

Inkrementiert das x-Register.
(LET X = X + 1)

iny

Inkrementiert das y-Register.
(LET Y = Y + 1)

Unter Dekrementieren versteht man etwas ganz ähnliches, hier wird im Gegensatz zum Inkrementieren der Wert des zu behandelnden Datums um Eins heruntergesetzt.

dec $nn (POKE $nn, PEEK ($nn) - 1)

```
dex         (LET X = X - 1)

dey         (LET Y = Y - 1)
```

Auch hier werden unter Umständen die Flags für Zero und Negativ gestzt.

2. Gruppe: Befehle zur Bitmanipulation

Hierbei handelt es sich um Befehle, mit denen Operationen der Binären Arithmetik ausgeführt werden können. Die Funktion sollte Ihnen spätestens seit dem Kapitel Binäre Arithmetik bekannt sein.

Sämtliche Befehle veranlassen eine Verknüpfung des Akkumulators mit dem angegebenen Wert bzw. dem Wert unter der angegebenen Adresse.

Die Syntax entspricht dem bisher gesehenen und auch hier werden die Zero- und Negativ-Flags beeinflußt.

```
and $nn     (A = A AND PEEK ($nn))

and #$n     (A = A AND $n)


eor $nn     (A = A EOR PEEK ($nn))

eor #$n     (A = A EOR $n)


ora $nn     (A = A OR PEEK ($nn))

ora #$      (A = A OR $n)
```

Damit wäre schon fast alles über diese Befehlsgruppe gesagt, machen wir also mit der nächsten Gruppe weiter.

3. Gruppe: Vergleichsbefehle

Auch bei der Maschinensprache gibt es die Möglichkeit, den Programmablauf auf bestimmte Ereignisse hin zu überwachen. Dies geschieht mit den COMPARE-Befehlen. Sie bewirken folgendes: Das angegebene Register wird dem angegeben Wert geANDet, das Ergebnis jedoch nicht festgehalten, sondern lediglich die Flags im Status-Register werden verändert.

```
cmp $nn      vergleicht den Akku mit der Adresse $nn

cmp #$n      vergleicht den Akku mit dem Wert $n


cpx $nn      vergleicht das x-Register mit der Adresse $nn

cpx #$n      vergleicht das x-Register mit dem Wert $n


cpy $nn      vergleicht das y-Register mit der Adresse $nn

cpy #$n      vergleicht das y-Register mit dem Wert $n
```

Wenn Sie sich jetzt fragen werden, wie man Nutzen aus den Vergleichen ziehen kann, haben wir schon einen Übergang zu den Sprungbefehlen gefunden, ohne die die COMPARE-Befehle relativ witzlos wären.

4. Gruppe: Sprungbefehle

Es gibt zwei Gruppen von Sprungbefehlen, die bedingten, d.h. an eine Bedingung geknüpfte Sprünge und unbedingte Sprünge.

Bedingte Sprünge

Im Basic wird ein bedingter Sprung durch eine IF ... THEN Anweisung erreicht.

In Maschinensprache geschieht das auf ähnliche Weise, lediglich die Auswahl der Bedingungen ist nicht so vielfältig. Hier klärt sich dann auch die Bedeutung der Status-Flags auf. Ein BRANCH-Befehl testet ein bestimmtes Flag und verzweigt dann. Bei nicht gegebener Bedingung wird der nächste Befehl ausgeführt, der Sprung also ignoriert. Bei wahrer Bedingung wird es ein bißchen komplizierter: Jeder BRANCH-Befehl besteht aus 2 Bytes. Das erste gibt die Bedingung an, bei der gesprungen wird, das zweite, wie weit der Sprung ist und die Richtung (nach "oben" oder "unten"). Es handelt sich bei den BRANCH-Befehlen also um relative Sprünge. Der Sprungbereich deckt die Zone von -126 bis +129 Bytes um den eigentlichen Befehl ab. Bei einem Sprung nach vorne muß das zweite Byte einen Wert zwischen 0 und 127 besitzen. Die Länge des Sprunges wird durch den Wert des Bytes angegeben. Der "Absprungpunkt" ist die dem BRANCH-Befehl unmittelbar folgende Adresse.

Ein Sprung rückwärts wird durch den Wert 128 bis 255 erreicht. Die Sprunglänge errechnet sich aus: 256 - Wert des Bytes.

```
beq        verzweigt bei gesetztem Zero-Flag

bne        verzweigt bei nicht gesetztem Zero-Flag

bmi        verzweigt bei gesetztem Negativ-Flag
```

bpl	verzweigt bei nicht gesetztem Negativ-Flag

Unbedingte Sprünge

Unbedingte Sprünge kennen wir vom Basic her als GOTO bzw. GOSUB.

Hier gibt es bei der Maschinensprache natürlich ein Äquivalent, ohne das ein Programmieren ja wohl auch kaum möglich wäre.

jmp $nn	springt nach $nn (GOTO $nn)
jsr $nn	springt in die Unterroutine ab $nn (GOSUB $nn)
rts	entspricht dem Basic-Befehl RETURN

5. Gruppe: Befehle zur Prozessorsteuerung

Mit Hilfe dieser Befehle wird dem Prozessor ein bestimmter Modus mitgeteilt.

sed	schaltet den BCD- (Binär Codierte Dezimalzahl) Modus ein
cld	schaltet den BCD-Modus ab
clc	löscht das Carry-Flag
nop	keine Bedeutung, Platzhalter

So, das sind die vorkommenden Maschinensprachebefehle, eine zwar nicht vollständige Liste, doch für unsere Zwecke reicht es erst einmal aus. Um die Maschinensprache zu lernen, ist es zweifellos zuwenig. Wer sich mit diesem interessanten Gebiet der Programmierung beschäftigen will, dem ist unbedingt anzuraten, sich mit entsprechender Fachliteratur einzudecken. Ein Schritt, der sich zweifellos lohnt.

12. Interrupt-Erweiterungen

Eine sehr bequeme und elegante Art, bewegliche Objekte auf den Bildschirm und vor allem in Bewegung zu bringen, ist es, die Interrupt-Routine zu erweitern.

Die Vorteile der Technik:

Kein Zeitverlust für die Abfrage von Joystick, Bewegung von Sprites.

Schnelle Bewegungen sind einfach zu programmieren.

Gleichmäßige Bewegungen. Es gibt keine abgehackten Bewegungen, wie Sie sie von Basic-Programmen her kennen.

Man kann ohne Probleme auch 8 Sprites bewegen. Die Bewegungen bleiben flüssig und schnell.

Basic-Programme laufen mit fast der gleichen Geschwindigkeit ab, mit der sie ohne erweiterten Interrupt ablaufen würden (Bei einer Zählschleife von 1 bis 1000 mit Bildschirmausgabe der Zahl; inklusive Scrolling brauchte das Programm mit dem normalen Interrupt genauso lange (mit der Stoppuhr gemessen, nicht mit TI$) wie dasselbe Programm mit veränderter Interrupt-Routine, nur mit dem Unterschied, daß während des zweiten Durchgangs noch Sprites auf dem Bildschirm bewegt wurden).

Allerdings müssen so viele Vorteile auch mit einem Nachteil erkauft werden: Mit Basic läßt sich keine erweiterte Interrupt-Routine schreiben.

Nachteil:

Programmierung nur in Maschinensprache möglich.

Dieser Nachteil soll uns aber nicht abschrecken. Zur Steuerung von Sprites würde man in Basic mit POKEs arbeiten, so etwas ist aber auch in Maschinensprache sehr leicht zu machen.

12.1. Veränderung des Interrupts

Wie gehe ich vor, wenn ich mir eine eigene Interrupt-Routine schreiben will?

Dazu erst einmal folgendes: Es ist klar, daß ich den Interrupt nicht so verändern darf, daß er seine Aufgaben nicht mehr erfüllen kann. Der Versuch, eine völlig eigene Interrupt-Routine zu schreiben, also eine, die lediglich zur Steuerung von Sprites dient, dürfte wohl den Computer dazu veranlassen, in einen Streik zu treten. Wir werden also auch weiterhin nicht auf den normalen Interrupt verzichten können. (Wir haben ja auch kein Grund, so etwas zu wollen.)

Der Weg, den wir einschreiten, ist also folgender:

Wir schreiben uns eine Routine, die für uns, nur um ein Beispiel zu nennen, den Joystick abfragt, und nach der Stellung des Joysticks ein Sprite über den Bildschirm bewegt.

Als zweites leiten wir den Interrupt um. Wir teilen ihm einfach mit, daß er nicht mehr die normale Adresse anspringen soll, sondern erst z.B. die Adresse 49152. Dort steht dann unser Programm für die Sprite-Steuerung. Am Ende des Programms steht dann ein Sprungbefehl zu der normalen Interrupt-Routine. Das wäre dann die Adresse $EA31 (59953). Mit diesem Befehl wird der Interrupt also wieder in normale Bahnen geleitet.

12.2. Veränderung des Interrupt-Vektors

Der Interrupt-Vektor ist ein Pointer, der normalerweise auf die Adresse $EA31 zeigt. Dieser Pointer ist in den Adressen 788/789 zu finden. Auch hier hat also die Zero-Page mal wieder ihre Finger mit im Spiel.

Wollen wir den Pointer umändern, so sollte man darauf achten, daß nicht der Fall eintritt, daß genau zwischen der Veränderung des Low-Bytes und des High-Bytes ein Interrupt ausgelöst wird. Dieser Fall wird zwar sehr selten eintreten, wäre aber sehr peinlich, da der Rechner sich sofort aufhängen würde.

Es gibt zwei Möglichkeiten, diese Eventualität auszuschließen:

Man wartet einen Interrupt ab und verändert dann im nächsten Schritt den Pointer. So langsam ist Basic nun auch wiederum nicht, als daß man befürchten müßte, bis zum nächsten Interrupt nicht den gesamten Pointer verändert zu haben.

Eine Befehlsfolge, die diese Technik ausnutzt:

POKE 198, 0: WAIT 198, 1

Mit POKE 198, 0 wird die Länge des Tastaturpuffers auf Null gesetzt. Das bedeutet im Klartext, er wird gelöscht. WAIT 198, 1 wartet darauf, daß ein Zeichen in den Tastaturpuffer eingeschrieben wird, also eine Taste gedrückt wird. Da die Tastatur-Abfrage eine Aufgabe des Interrupts ist, können wir also sicher sein, daß in dem Moment, wo in der Speicherzelle 198 eine 1 auftritt, der Interrupt gerade "vorbei" ist, wir also 1/60 Sekunde Zeit haben, bevor der nächste ausgelöst wird.

Diese Zeit nutzen wir für die Veränderung des Pointers:

POKE 788, Low: POKE 789, High

Die zweite Möglichkeit ist rabiater, aber auch sicherer. Man schaltet den Interrupt ab, verändert den Pointer und schaltet den Interrupt wieder ein.

POKE 56334, PEEK (56334) AND 254

Mit diesem Befehl haben wir den Interrupt abgeschaltet.

Wir können jetzt auf bekannte Art und Weise den Pointer verändern.

Zum Schluß müssen wir allerdings den Interrupt wieder einschalten. Dies geschieht mit:

POKE 56334, PEEK (56334) OR 1

Es dürfte klar sein, daß beide Möglichkeiten nur innerhalb eines Programms verwendet werden können.

13. Tele-Spiele

Sieht man die heutige Vielfalt der Spielmöglichkeiten, so ist es erstaunlich, in welch kurzer Zeit die Entwicklung vom einfachen Tennis-Spiel bis zu den heutigen ausgefeilten Spielvarianten ablief.
Vor ca. 10 Jahren war es eine Sensation, wenn ein einfacher Punkt über den Bildschirm sauste, ein Spiel, das heute schon beinahe in Vergessenheit geraten ist. Nichtsdestoweniger erscheint es mir interessant, einmal den Weg der Entwicklung aufzuzeigen. Man hat die verschiedenen Spiele in Generationen eingeteilt, mit denen man den Fortschritt gut verdeutlichen kann.

Zu der 1. Generation zählt man Spiele wie das schon zitierte Tennis. Diese Spiele kamen mit einem Minimum an Graphik und Aufwand aus. Die 2. Generation fiel da schon komplizierter aus. Die Figuren wurden "feiner" und häuften sich, die Graphik überhaupt wurde beweglicher und komplizierter. Ein Beispiel aus dieser Generation wäre z.B. das Spiel "Panzer-Jagd". Die 3. Generation wurde mit dem legendären Spielhit "Pac Man" ins Leben gerufen. Fast alles was heute in Spielotheken zu finden ist, kann dieser Generation zugeordnet werden. Die 4. Generation ist sozusagen der letzte Schrei: Die zu dem Spiel gehörende Graphik ist auf einer Laser-Bildplatte gespeichert und derart ausgefeilt, daß man sie kaum von einem Fernsehbild unterscheiden kann; eine Technik, die zumindest vorerst den Hobbyanwendern verschlossen bleibt - denn wer hat schon die Möglichkeit, eine solche Bildplatte in sein System einzubeziehen, von der aufwendigen Programmierung und den hohen Kosten ganz zu schweigen.

Interessant für den Hobby-Anwender sind vor allem die Spiele der 1. bis 3. Generation. Hiermit wird sich das Buch beschäftigen.

Die heutige Vielfalt von Tele-Spielen macht es unmöglich, diese alle unter einen Hut zu bringen. Wie sollte man auch ein Spiel wie "Pac Man" mit z.B. "Phoenix" vergleichen, in denen vom Spieler völlig verschiedene Verhaltensweisen abverlangt werden. Während er bei "Phoenix" gleich einem Westernheld wild herumballert, um sich die Masse der "Angreifer" vom Leibe zu halten, muß er beim friedlichen "Pac Man" Kekse knabbernd durch ein Labyrinth irren, immer auf der Hut vor Gespenstern, und kann sich seinen Kontrahenten meist nur durch eine mehr oder weniger kopflose Flucht entziehen.

Für uns stellt sich also das Problem, wie wir uns durch diesen Wust der nahezu unendlichen Möglichkeiten "durchbeißen", ohne unser Ziel aus den Augen zu verlieren.

Glücklicherweise weisen viele Programme immer wiederkehrende Parallelen auf. Ich werde also versuchen, Ihnen einige dieser immer wieder auftretenden Situationen zu zeigen. Dies wird größtenteils mit Hilfe eines Beispielprogrammes geschehen, teilweise aber auch theoretisch, z.B. wie man eine vorhandene Routine erweitern kann, so daß sie schließlich ganz andere Aufgaben erfüllen kann. Dies wird am Ende dieses Teiles geschehen, hier ist also eine gewisse Mitarbeit gefragt. Ich glaube aber, daß ich Sie auch an dieser Stelle nicht überfordern werde.

Durch die ähnlichen Situationen, die in den verschiedenen Programmen auftauchen, liegt es auf der Hand, Spielprogramme in Kategorien zu unterteilen.

Schauen wir uns zuerst einmal das schon öfters zitierte Spiel "Pac Man" an: Ohne sich mit der Frage des WIE zu belasten, schauen wir uns nur an, WAS passiert, d.h. den einfachen Spielablauf.

Das Spielfeld ist ein Labyrinth, das sich im Laufe des Spiels weder verändert noch bewegt. Das Labyrinth dient dazu, die freie Beweglichkeit der Figuren einzuschränken und

hat ansonsten eigentlich wenig Funktionen. Es hat zum Beispiel wenig Folgen, wenn eine Spielfigur an eine Mauer stößt, sie wird zwar aufgehalten, darf dafür aber auch weiterspielen.

Die Spielfiguren können sich, soweit es die Mauern erlauben, frei auf dem Bildschirm bewegen.

Dies sind meiner Meinung nach Kriterien, die schon eine recht gute Einordnung zulassen. Insgesamt unterscheiden wir sechs Kategorien:

1. Kategorie: Spiele mit festem Hintergrund und frei beweglichen Spielfiguren. (An einem solchen Beispiel möchte ich die allgemeine Genese von Tele-Spielen erläutern.)
Nehmen wir uns nun das nächste Spiel vor, "Phoenix":
Hier hat der Spieler ein Raumschiff, das er am unteren Rand des Bildschirms hin und her bewegen kann. Des Spielers Gegner dagegen können sich auf dem gesamten Bildschirm bewegen.

2. Kategorie: Fester Hintergrund, eigene Spielfigur läßt sich nicht frei bewegen, die "Gegner" sind frei beweglich.
Mit diesen beiden Gattungen können wir schon viele Spiele einordnen, de übrigen vier lauten wie folgt:

3. Kategorie: Beweglicher Hintergrund, eigene Figur läßt sich nicht frei bewegen (z.B. "Pole Position").

4. Kategorie: Beweglicher Hintergrund, eigene Figur läßt sich frei bewegen (z.B. "Defender"). Nachdem die ersten drei Kategorien sehr ausführlich behandelt werden, wird diese Kategorie mit Hilfe der anderen abgehandelt.

5. Kategorie: Adventures (z.B. "Hobbit"). Hierüber möchte ich mich dann nicht mehr auslassen, da es hier schon Bücher gibt, und das Gebiet zu weitreichend ist, um es in den hier zu Verfügung stehenden Platz zu zwängen.

6. Kategorie: Strategiespiele (z.B. Schach). Auch so etwas werden Sie vergeblich suchen. Dieses Thema ist meiner Meinung nach viel zu kompliziert, als daß man Ergebnisse erwarten darf, die in einer Relation zu der damit verbundenen Arbeit stehen.

14. Basic für Telespiele

Die ersten Versuche, ein Tele-Spiel zu schreiben, werden wohl fast immer in Basic unternommen. Dies liegt zum einen daran, daß Basic die am weitesten verbreitete höhere Programmiersprache ist, andererseits, daß Basic leicht erlernbar ist, und über viele Befehle verfügt, die sich für Spiele anbieten. Nun sollte sich die Frage stellen, warum diese Vorteile nicht ausreichen, um interessante Spiele zu programmieren. Die Antwort vorweggenommen: Basic hat neben seinen Vorteilen einen ganz entscheidenden Nachteil: Die Sprache ist relativ langsam. Für Spiele ist es aber wichtig, daß viele Aktionen scheinbar gleichzeitig ablaufen. Da müssen Positionen berechnet, Bilder auf dem Bildschirm bewegt, Kollisionen überwacht und ausgewertet werden, und zu guter Letzt soll ja ein Spiel von Geräuschen begleitet sein. Verdeutlichen möchte ich dieses Problem anhand eines Spieles, das aus der ersten Generation stammt: Pelota. Es geht dabei darum, einen Ball immer wieder gegen eine Mauer zu schlagen (das dies nur eine sehr grobe Wiedergabe des wirklichen Spiels ist, dürfte klar sein). Im Spiel selbst gibt es nur zwei bewegliche Objekte, den Ball und einen Schläger, und auch keine "Geräuschkulisse", trotzdem erweist sich Basic als fast schon zu langsam. Schauen wir uns nun das Spiel an.

Listing: Pelota

```
10 PRINT"□"
20 PRINT"▣
          ";
30 FORI=0TO22
40 PRINT"▣ ▀
          █";:NEXTI
50 PRINT"▣
         ";
60 POKE2023,160:POKE650,128
70 FORI=55296TO56295:POKEI,0:NEXTI:A=0:T
I$="000000"
80 YS=13:IX=1:IY=1:Y=2:X=INT(RND(1)*22)+
2
90 POKE1024+36+YS*40,160
100 POKE1020+YS*40,160
110 POKE1100+YS*40,160
120 GETA$:GOSUB190
130 IFA$<>""ANDA$<>""THEN90
140 IFA$=""THENYS=YS-1:POKE1140+YS*40,32
:GOTO170
150 YS=YS+1:POKE980+YS*40,32:IFYS>22THEN
YS=22
160 GOTO90
170 IFYS<2THENYS=2
180 GOTO90
190 IFX<2ORX>22THENIX=-IX
200 IFY<2THENIY=-IY
210 IFY=37THEN270
220 IFY>34ANDPEEK(1024+40*X+Y)=160THENIY
=-IY
230 POKE1024+40*X+Y,32
```

```
240 X=X+IX:Y=Y+IY
250 POKE1024+40*X+Y,81
260 RETURN
270 POKE1024+36+YS*40,32
280 POKE1020+YS*40,32
290 POKE1100+YS*40,32
300 POKE1024+40*X+Y,32:A=A+1
310 PRINT"[illegible]    FEHLER:";A
320 IFA>14THEN340
330 GOTO80
340 PRINT"[illegible]";
350 PRINT"DAS SPIEL DAUERTE:";TI$;" MIN/
SEC."
360 PRINT"                          NOCHMAL (J/N)?
"
370 GETQ$:IFQ$=""THEN370
380 IFQ$="J"THENRUN
390 END
```

Was geschieht nun im einzelnen?

10:	Löschen des Bildschirms
20-50:	Zeichnen eines Rahmens
60:	Letzter Punkt des Rahmens & Repeat on
70:	Farbram initialisieren
80:	Variablen initialisieren
90-110:	Schläger POKEn (YS= Schlägerposition)
120:	Tastaturabfrage
130-140,160:	Kollision mit der Wand?
150:	Ball im Aus?
170:	Löschen des Balls
180:	Neue Koordinaten des Balls
190:	Setzen des Balls
200-220:	Auswertung der Tastaturabfrage
230:	Neuer Durchlauf
260-290:	Löschen des Balls & des Schlägers
300:	Fehleranzeige
310:	Neuer Ball?
320:	Neuer Ball
340:	Auslesen der Uhr
350:	Bewertung
360:	Neues Spiel?
370:	Tastaturabfrage
380:	Neues Spiel
390:	Ende

Für jemanden, der sich bereits ein wenig mit Programmieren beschäftigt hat, sollte das Programm jetzt klar sein. Anfänger haben vielleicht noch ein paar Fragen und diese möchte ich jetzt klären. Das erste, worauf ich näher eingehen möchte, ist die Zeile 50: Jeder weiß, daß der C-64 mit 40 Zeichen pro Zeile arbeitet. Trotzdem hat der untere Balken des Rahmens zunächst einmal nur eine Länge von 39 Zeichen, und das mit gutem Grund: Wäre diese Zeile um ein Zeichen länger, so würde der gesamte Rahmen um eine Zeile nach oben geschoben, also zerstört. Verantwortlich dafür ist das sogenannte "Scrolling". Es sorgt dafür, daß auf dem Bildschirm immer genügend Platz für weitere Eingaben ist.

Diese an sich sehr nützliche Einrichtung muß hier jedoch überlistet werden. Dies geschieht zum einen dadurch, daß, wie bereits erwähnt, das letzte Zeichen ausgelassen wird. Nun wollen wir aber den Rahmen geschlossen haben. Das letzte Zeichen müssen wir also unter Umgehung des PRINT-Befehls erzeugen. Dies geschieht nun in der Zeile 60: POKE 2023,160: "Schreibe in die Speicherzelle 2023 den Wert 160 (die Syntax des POKEs lautet: POKE Adresse,Byte)", nun verkörpert die Speicherzelle 2023 die rechte untere Ecke des Bildschirms, ist also das "Loch" in unserem Rahmen. Der Wert 160 ist nichts anderes als der Bildschirmcode für das reverse Space. Dieser POKE allein nützt jedoch wenig, da das Zeichen vorerst die Farbe des Bildschirmhintergrundes annimmt, also unsichtbar bleibt. Die Farbe der Zeichen auf dem Bildschirm, die im RAM-Bereich von 1024 bis 2023 stehen, nimmt sich der Rechner aus dem Bereich von 55296 bis 56295, daher die FOR-NEXT-Schleife in Zeile 70. Hier wird jeweils der Wert 0 eingePOKEt. Er steht für die Farbe Schwarz. Der Grund, warum die Farbe nicht nur an der einen Stelle gesetzt wird, ist folgender: Beim Ablauf des Programms wird der springende Ball ebenfalls durch POKEs erzeugt. Setzt man die Farbe, die ja gleich bleibt, schon im voraus, so kann man Erhebliches an Zeit sparen.

Das POKE kann aber noch anderes: Durch POKE 650,128 in Zeile 60 wird bewirkt, das sämtliche Tasten solange wiederholt werden, wie sie gedrückt bleiben. Man sieht also, daß der POKE-Befehl recht nützlich ist. Er ist aber auch "gefährlich": Der Computer kann ein falsches POKE einem sogar recht übelnehmen. Nicht, daß er davon zerstört würde, nein, er läßt einfach nicht mehr mit sich reden. Versuchen Sie einmal folgendes (aber bitte nur, wenn keine wichtigen Daten oder Programme im Speicher sind!): POKE 1,0. Ergebnis: Nichts geht mehr. Sie können jetzt versuchen was Sie wollen, es wird Ihnen nicht mehr gelingen, mit dem Rechner irgend etwas Vernünftiges anzufangen. Aus dieser Misere hilft nur eins: Schalten Sie den Rechner aus und wieder ein - schon ist er wieder bereit, Ihnen zu Diensten zu sein.

Die restlichen POKEs im Programm dienen dazu, den Ball oder den Schläger zu bewegen, z.B. POKE 1024+40*X+Y,81. Hierbei bedeutet: X die Spalte, Y die Zeile und der Wert 81 entspricht dem Ball.

Der nächste Befehl, der zu Fragen führen kann, steht in Zeile 120: GET A$. Hier wird nichts anderes gemacht, als die Tastatur abgefragt. Das heißt im Klartext: Wenn in dem Moment, in dem der Rechner diesen Befehl abarbeitet, eine Taste gedrückt ist, so wird das Zeichen, das dieser Taste zugeordnet ist, in die Variable A$ eingegeben. Dieser Befehl ist z.B. wichtig, wenn man Figuren oder ähnliches über Tasten steuern will.

Weitere Probleme könnte wohl die Zeile 340 aufwerfen. Hier wird die interne Uhr des C-64 ausgelesen, die als "Punktezähler" des Programmes dient. Die Uhr selber wird durch die Variable TI$ verkörpert. TI$ ist eine Systemvariable, die "automatisch" im Sekundentakt aufgezählt wird. Sie ist, wie das Dollarzeichen schon zeigt, eine String- oder Zeichenvariable. Ihre Länge beträgt immer 6 Zeichen des Formates "Std.Std.Min.Min.Sec.Sec.". Für das Programm sind nur die letzten vier Stellen, d.h. die Minuten und Sekunden, interessant. Durch die Funktion MID$ in der besagten Zeile werden die mittleren beiden Zeichen, also die Minuten der Variable M$ zugeordnet. Die Sekunden werden ähnlich ausgefiltert. Hier habe ich mich allerdings der Funktion RIGHT$ bedient.
So, nunmehr dürften alle Probleme, die durch das Programm aufgetreten sein könnten, geklärt bzw. mit Hilfe des Handbuches zu lösen sein.

Man sieht dem Spiel an, daß man einem Basic-Programm nicht mehr als diese wenigen bewegten Objekte zumuten darf. Es dient also quasi zur Begründung, warum ich Sie mit der Maschinensprache konfrontiert habe.

TEIL 3

Kommen wir endlich zu den Schritten, die für ein Spiel vonnöten sind.

14.1. Die Spielidee

Bevor man ein Spiel schreiben kann, muß man sich Gedanken darüber machen, wie das Endergebnis aussehen soll.

Diese vorerst rein theoretischen Überlegungen möchte ich einmal unter dem Begriff Spielidee zusammenfassen.

Versucht man, sich an den Rechner zu setzen und einfach darauf los zu programmieren, so wird man schnell in irgendeiner Sackgasse landen und resigniert erst einmal aufgeben.

Bevor man den Rechner zum ersten Mal einschaltet, sollte das Spiel auf dem Papier schon konkrete Formen angenommen haben.

Es reicht nicht, wenn man sagt: Ich programmiere jetzt Pac Man, sondern man sollte sich die Mühe machen, jede Routine genau zu definieren.

Schreiben Sie alle Überlegungen auf. Sie werden staunen wieviel da zusammenkommt. Allenfalls zuviel, um es sich merken zu können.

Spielen Sie das Spiel in Gedanken durch. Nur so können Sie sicher gehen, alle Einzelheiten festzuhalten.

Schreiben Sie dazu auf:

- Was ist die Aufgabe des Spielers?
- In welchem Hintergrund bewegt sich das Spiel?
- Wie sollen die Figuren aussehen?
- Wie viele Figuren gibt es?
- Gibt es mehrere Bilder? Wenn ja, wie gelangt man ins nächste?
- Welche Hindernisse liegen dem Spieler im Weg?
- Wie kann man den Hindernissen aus dem Weg gehen, sie beseitigen.
- Wofür gibt es wieviel Punkte?
- Wie soll das Spiel untermalt sein. Soll es eine Hintergrundmusik geben?
- Welches Lied soll gespielt werden?
- Gibt es sonstige Geräusche?

Haben Sie diese Fragen beantwortet, so müssen Sie Bilanz ziehen. Ist das Spiel ausreichend abgegrenzt? Oder gibt es noch Situationen, die Sie nicht mit Hilfe der obigen Liste von Beispielfragen überschauen können? Diese Phase der Planung darf erst dann abgeschlossen sein, wenn Sie in der Lage sind, das gesamte Spiel zu überblicken.

Sie müssen zu jeder Situation eine Antwort haben. Einige solcher Situationen möchte ich einmal am Beispiel Pac Man erläutern.

Wie wird die Spielfigur gesteuert? In welche Richtungen kann sie gesteuert werden? (Antwort: Mit dem Joystick, in alle vier Richtungen.)

Wie frei läßt sich die Spielfigur bewegen? Gibt es irgendwelche Einschränkungen? (Der Bildschirm zeigt ein Labyrinth. In den Gängen des Labyrinthes läßt sich die Spielfigur an jede Position des Bildschirms bewegen.)

Was passiert, wenn die Spielfigur an den Hintergrund stößt? (Sie wird angehalten, kann den Weg also nicht weiter fortsetzen.)

Wie sieht das Labyrinth aus? Gibt es Sackgassen? (Zeichnung des Labyrinths)

Was muß der Spieler im Labyrinth tun? (In den Gängen sind Punkte verteilt, der Spieler muß mit seiner Figur diese Punkte aufsammeln.)

Wann ist die Aufgabe erfüllt? (Wenn alle Punkte aufgesammelt sind.)

Wie stellt das Programm fest, daß alle Punkte aufgesammelt worden sind? (Das Programm besitzt einen Zähler für die bereits aufgesammelten Punkten.)

Was passiert, wenn die Aufgabe erfüllt wurde? (Das Spiel beginnt von vorne, wird aber schwieriger)

Wie sammelt der Spieler die Punkte ein? (Es genügt, mit der Spielfigur über die Punkte zu laufen. Dann werden die Punkte gelöscht und der Punktestand des Spielers aufaddiert.)

Wird der Spieler durch irgend etwas an seiner Aufgabe gehindert? (Im Labyrinth bewegen sich noch andere Figuren. Berührt der Spieler einen dieser Verfolger, so hat er seine Spielfigur verloren.)

Wieviel Spielfiguren besitzt der Spieler? (Beispielsweise drei Stück.)

Wie kann der Spieler den Hindernissen ausweichen? (Er kann vor ihnen flüchten und in gewissen Situationen kann er die Verfolger auch fressen.)

Wann tritt eine Situation ein, in der der Spieler die Verfolger fressen kann? (auf dem Bildschirm gibt es vier besonders gekennzeichnete Punkte. Sammelt der Spieler einen dieser Punkte ein, so "stirbt" der Verfolger, wenn sich zwei Figuren berühren.)

Was geschieht mit den "gestorbenen" Verfolgern? (Sie kommen nach kurzer Zeit wieder ins Spiel und spielen dann wieder mit.)

Mit diesen Fragen ist in etwa festgelegt, was im Spiel passieren soll.

Sie haben vielleicht das System entdeckt, nach dem die Fragen aufgebaut wurden. Aus der Antwort der ersten Frage entsteht die zweite und so weiter. Beherzigt man dieses System, so kann man sich ziemlich sicher sein, alle Eventualitäten bedacht zu haben. Erst wenn sich aus irgendeiner Antwort keine neue Frage mehr bilden läßt, sollte man zu einer neuen Frage einwerfen.

Mit diesem übrigens auch in der Psychologie verwendeten Prinzip, dem "aktiven Zuhören", müßte es Ihnen möglich sein, alle Möglichkeiten zu bedenken.

14.2. Überprüfung auf Durchführbarkeit

Jetzt sollten Sie sich die Arbeit machen und überprüfen, wieviele Sprites zu welcher Zeit auf dem Bildschirm sind. Sind es mehr als 8? Dann haben Sie leider das Spiel zu kompliziert geplant. Die Aufgabe dieses Schrittes ist also die Überprüfung, ob alle Spielzustände ·mit unseren Mitteln (also etwa mit der begrenzte Anzahl von Sprites) durchgeführt werden können.

Dieser Schritt ist von ganz wesentlicher Bedeutung. Ich garantiere Ihnen, Sie wereden sich schwarz ärgern, wenn Sie mitten im Programm feststellen müssen, alle Arbeit war umsonst, weil es eine Situation zwingend erfordert, ein neuntes Sprite auf den Bildschirm zu bringen. Pac Man wird Sie in diese Verlegenheit nicht bringen, bei einem Weltraumspiel kann das schnell geschehen.

14.3. Überwachung der Spielzustände

Es ist nun an der Zeit, sich Gedanken darüber zu machen, wie Sie das Spiel überwachen wollen. Sie müssen jetzt eine Liste der Kriterien zusammenstellen, wie das Programm feststellen soll, ob eine Situation eingetreten ist, die eine gesonderte Bearbeitung erforderlich macht.

In den meisten Fällen wird eine Kontrolle des Spiels am besten durch die Überwachung von Zusammenstössen gelöst.

Für Pac Man sieht die Auflistung der routinemäßigen Aufgaben des Programms so aus:

- Joystick-Abfrage
- Bewegung der Spiefigur
- Bewegung der Verfolger
- Hintergrundmusik
- Abfrage der Kollisions-Register

Bei der Abfrage der Kollisions-Register können zwei Zustände festgestellt werden:

1. Es gibt keine Kollisionen.

Wird dies festgestellt, so ist die Aufgabe der Routine erfüllt, das Hauptprogramm kann wiederholt werden.

2. Es gibt Kollisionen.

Ist eine Kollision eingetreten, ist eine gesonderte Behandlung für jeden Fall angebracht.

Als erstes muß dazu festgestellt werden, um was für eine Kollision es sich handelt. Auch hier gibt es zwei Hauptgruppen:

Sprite-Sprite-Kollision

Es ergeben sich zwei grundsätzliche Möglichkeiten, ausgehend von der Frage: Ist unsere Spielfigur an der Kollision beteiligt?

- Ja: Spielfigur ist beteiligt.

Befinden wir uns in einer Situation, in der die Verfolger gefressen werden können?

- JA:

Löschen des Verfolgers. Wenn es jetzt keine Kollisionen mehr gibt, so kann man ins Hauptprogramm zurückkehren.

- Nein:

Wir haben soeben unsere Spielfigur verloren und müssen alle für diese Situation vorgesehenen Schritte einleiten.

- Nein: Spielfigur ist nicht beteiligt.

Es handelt sich also um einen Zusammenstoß zweier oder mehrer Verfolger.

Wir müssen dafür sorgen, daß die Verfolger getrennt werden, da es sonst passieren kann, daß die Verfolger "zusammenkleben". Der Grund für dieses denkbare Verhalten:

Löschen wir lediglich das Kollisionsregister, ohne die Sprites zu trennen, wird auch bei der nächsten Abfrage eine Kollision ausgelöst. Der mögliche Erfolg: Das Programm hängt sich auf, weil es in einer Endlosschleife gelandet ist.

Sprite-Hintergrundkollision

Auch hier müssen wir zuerst unterscheiden, welche Figur den Zusammenstoß verursacht hat.

- Spielfigur/Zusammenstoß mit Punkt

Der Punkt wird gelöscht, der Punktestand aufaddiert. Danach muß geprüft werden, um was für einen Punkt es sich gehandelt hat. Ging es um einen der vier Punkte, nach deren Genuß die Spielfigur Jagd auf die Verfolger machen kann, so muß dies dem Programm mitgeteilt werden.

- Spielfigur/Zusammenstoß mit Mauer:

Hier muß die Figur von der Mauer getrennt werden, ansonsten riskiert man wiederum das Aufhängen des Programms.

- Verfolger/Zusammenstoß mit Punkt:

Dieser Vorfall muß nicht weiter verfolgt werden, da die Punkte für die Verfolger keinerlei Bedeutung haben.

Zusammenstoß mit Mauer

Hier gilt das gleiche wie für die Spielfigur.

14.4. Die Programmierung

Die bisherige Beschäftigung war mehr oder minder theoretisch. Jetzt gilt es, die Idee in die Tat umzusetzen.

Normalerweise beginnt man mit den statischen Programmteilen. Man programmiert also zuerst die Sachen, die während des Programmablaufs nicht oder kaum verändert werden. In unserem Beispiel wäre das zum Beispiel das Labyrinth. Am einfachsten läßt es sich durch ein kurzes Basic-Programm erstellen. Hier wird dann mit Hilfe der vorgegebenen Zeichen ein Labyrinth mitsamt aller Punkte auf den Bildschirm gePRINTet.

Als weitere Schritte kommen dann die beweglichen Objekte dazu.

Zuerst die Spielfigur; kann sie alle Aufgaben erfüllen (bei diesem Spiel wäre das der Fall, wenn sich die Figur auf dem Bildschirm bewegen, dabei durch den Joystick gesteuert werden kann, Punkte frißt und Mauern als Hindernis erkennt), dann sind die anderen Figuren an der Reihe.

Immer wenn Sie einen Teil fertiggestellt haben, möchte ich Ihnen anraten, diesen Teil erst abzuspeichern, bevor Sie ihn testen. Gerade bei den Bewegungsroutinen, die ja in Maschinensprache geschrieben sein werden, kann es sehr leicht passieren, daß sich ein Fehler eingeschlichen hat. Je nach Fehler kann das aber unter Umständen den Verlust des Programms bedeuten.

14.5. Beispielprogramme

Auf den folgenden Seiten sind 4 mehr oder minder große Beispielprogramme aufgelistet und auch erklärt. In kurzen Zügen wird die Spielidee erläutert und verdeutlicht.

Das erste Programm in dieser Reihe nimmt dabei eine Sonderstellung ein, da es ohne Maschinensprache-Routine ist. Zu den anderen Programmen gibt es zu den Basic-Ladeprogrammen zusätzlich noch ein Assemblerlisting der verwendeten Maschinensprache-Routinen.

Wenn Sie noch nicht Maschinensprache programmieren können, so können Sie diese Routinen in eigene Programme übernehmen. Ich habe versucht, sie so zu schreiben, daß sie auch anderweitig zu verwenden sind.

14.5.1. Labyrinth

Das Spiel "Labyrinth" ist ein in Basic gehaltenes Spiel. Die Aufgabe des Spielers ist, durch ein Labyrinth zu finden. Es handelt sich also um ein Spiel mit festem Hintergrund und beweglicher Spielfigur.

Die Spielidee

Die Idee ist ebenso einfach wie reizvoll. Auf dem Bildschirm wird ein Irrgarten gezeichnet, die einzige Aufgabe des Spielers ist es, durch den Irrgarten zu finden.

Zur Ausführung ist folgendes zu sagen. Wäre im Spiel nur ein

Irrgarten abgespeichert, so würde das Spiel schnell sehr langweilig werden. Das Problem ist also, einen Weg zu finden, das Spiel so zu schreiben, daß nicht nur ein Irrgarten abgespeichert ist, sondern möglichst viele.

Das zu erreichen, ist das eigentliche Problem dieses Spiels.

Wie bei anderen Problemen, gibt es auch hier mehrere Wege, die zu dem gewünschten Ziel führen. Ich möchte hier drei mögliche Lösungen aufführen.

1. Weg: Ein Labyrinth "Generator"

Dies ist die aufwendigste der möglichen Lösungen. Man schreibt ein Programm, das in der Lage ist, einen Irrgarten zu zeichnen. Das klingt noch ganz einfach, ist aber bei näherer Betrachtung ungemein schwierig. Das Problem ist weniger die Zeichnung eines Irrgarten mit einem gültigen Weg von A nach B, als das Problem, diesen Irrgarten möglichst schwierig zu gestalten. Ein einfach gehaltenes Programm würde wohl ein Labyrinth zeichnen, aber man kann davon ausgehen, daß es sehr leicht ist, einen Weg durch die Gänge zu finden.

2. Weg: Mehrere komplette Irrgärten

Sehr aufwendig ist die Technik, mehrere komplette Irrgärten abzuspeichern. Dieser Weg besticht durch seine Plumpheit sowie durch den immensen Speicherplatzbedarf. Es ist also ein echtes Negativbeispiel. Zu diesen Nachteilen gesellt sich auch noch die Tatsache, daß die Forderung nach vielen verschiedenen Labyrinthen nicht erfüllt wird. Selbst ein Rechner wie der C-64 würde unter dem immensen Speicherplatzbedarf regelrecht zusammenbrechen.

3. Weg: Zusammengesetzte Labyrinthe

Der meiner Meinung nach ideale Weg ist die Verwendung von einer begrenzten Anzahl von Labyrinth-Fragmenten, von denen erst eine gewisse Anzahl, in diesem Fall 4, ein bildfüllendes Labyrinth ergeben. In dem unten angegebenen Listing werden Sie die Informationen für 8 dieser Fragmente finden. Wie gesagt, sollen vier davon ein komplettes Labyrinth ergeben. Um aber zu gewährleisten, daß es immer einen gültigen Weg gibt, müssen die Fragmente einer gewissen Norm unterworfen sein. Sie müssen so entworfen sein, daß 2 beliebige Teile zusammenpassen und einen gültigen Weg ergeben.

Die hier verwendeten Teile haben ein gemeinsames Merkmal: Betrachtet man nur den äußeren Rand, so wird man feststellen, daß die Durchbrüche durch die umgebende Mauer bei allen Teilen deckungsgleich sind.

Alle Teile sind von einem Ring folgender Form umgeben:

```
xxxRxxxAxxxxxxExxxxx
x                  x
x                  x
R                  R
x                  x
x                  x
x                  x
E                  A
x                  x
x                  x
A                  E
xxxRxxxExxxxxxAxxxxx
```

Bei der Skizze bedeutet:

R: Richtiger Weg.

A: Ausgang. Hier führt eine Sackgasse in ein anderes Teil. Es ist zu beachten, daß sich jeweils ein A und ein E gegenüber liegen.

E: Eingang. Dieser Weg ist eine Sackgasse.

Innerhalb des oben gezeigten Ringes kann nun jeder Teil nach Belieben ausgefüllt sein. Lediglich die folgenden beiden Punkte sind zu beachten: Von R nach R muß es einen Weg geben, es darf keinen Weg von E nach R geben.

Sie werden sich vielleicht fragen, wie man auf so eine Lösung kommt: Purer Zufall. Mir ist sie beim Puzzlen eingefallen.

```
10 PRINT"□";
20 PRINT"************************************
*********";
30 PRINT"*******************LABYRINTH*****
************";
40 PRINT"************************************
*********"
50 PRINT"DIE AUFGABE IST ES, DASS ABGEBI
LDETE LA-"
60 PRINT"BYRINTH IN MOEGLICHST KURZER ZE
IT, UNTER    "
70 PRINT"VERWENDUNG  VON MOEGLICHST WENI
G ZUEGEN,     "
80 PRINT"ZU DURCHQUEREN. DAS 'Z' IST DAS
 ZIEL DIE     "
90 PRINT"AUGENBLICKLICHE LAGE IST DURCH
'' GE-       "
100 PRINT"KENNZEICHNET. ZUR STEUERUNG BI
TTE JOY-     "
110 PRINT"STICK 2 BENUTZEN; VIEL SPASS "
120 FORI=0TO3000:NEXTI
```

```
200 DATA"XXX XXX XXXXXX XXXXX","X   X X 
XX   X     X"
210 DATA"XXX        XXXXXXXXX","   XXX X
X  X        "
220 DATA"XX X   X X   XXXXX X","XX   XX 
 X XX    X X"
230 DATA"XXXX   X X  X XX X X","   X X X
 X X     X  "
240 DATA"XXX  X X X  XXXX X X","X   X  X
 X X   X XXX"
250 DATA"  XX  X  X   X X    ","XXX  XX 
XXXXXX XXXXX"
260 DATA"XXX XXX XXXXXX XXXXX","X X     
 X     X   X"
270 DATA"    X XXXXXX XX  X X"," XXXX   
  X X   X   "
280 DATA"X    XX X    XX  X X","X XX XX 
XX X   X  XX"
290 DATA"X    X   X  XX  X  X","  X X  X
 XX   X     "
300 DATA"XX X  X XX XX  X X X","XX XX X 
XX   X XX XX"
310 DATA"      X    XX   X   ","XXX XXX 
XXXXXX XXXXX"
320 DATA"XXX XXX XXXXXX XXXXX","X     X 
X   X   X  X"
330 DATA"XXX X X   X XXXXXX X","  X X XX
XXX     X   "
340 DATA"X  XX       X XXX XX","XX  XXXX
XX X  X    X"
350 DATA"X X      X XX X X XX","  X XX X
   XX X X   "
360 DATA"XX    X  XXX  X XXXX","X  XX  X
XX   X     X"
370 DATA"  X   X   X    X X X","XXX XXX 
XXXXXX XXXXX"
380 DATA"XXX XXX XXXXXX XXXXX","X X   X 
  X XX  X  X"
```

```
390 DATA"X XXX X X X   XXX  X","   XX   
X   X XX    "
400 DATA"XX XXXXX  XX  X X XX","XX   X  
 X X X  X  X"
410 DATA"XXXX X X X X  X X XX","X      X
XX XX   X   "
420 DATA"X XXXXXX     X  XX X","X      X
 X X XX    X"
430 DATA"  X XXX  X X   XX X ","XXX XXX 
XXXXXX XXXXX"
440 DATA"XXX XXX XXXXXX XXXXX","X   XX  
XXX XX     X"
450 DATA"XXX    X      XXXXXX","  XX X  
XX X X      "
460 DATA"X XXX         XX XXX","X  XX XX
X XX XX    X"
470 DATA"XX  X   X  XX X XX X","  X XXX 
 X       X  "
480 DATA"X X    X XX X XXXXXX","XX XXX  
 XX X    XXX"
490 DATA"       X   XXX XX   ","XXX XXXX
XXXXXX XXXXX"
500 DATA"XXX XXX XXXXXX XXXXX","XXX   X 
    X  XX  X"
510 DATA"XX XX XXXXX  XX   XX","   X  X 
X     X X   "
520 DATA"XX  X     X X   X XX","X X XXXX
XXXXXX X  XX"
530 DATA"X X          X X X X","  XXX XX
XX X X X    "
540 DATA"XXX   X   X  XX  X X","X X X X 
X XX    X XX"
550 DATA"    X X   XXXX X    ","XXX XXX 
XXXXXX XXXXX"
560 DATA"XXX XXX XXXXXXXXXXXX","X       
 X       XXX"
570 DATA"XXXX XXXX  XX XX   X","   X   X
  X   XX X  "
```

```
580 DATA"XX  XX X XX XX   XXX","XXX  X  
   XXX XXXXX"
590 DATA"XX  X  X X   X     X","  X   X 
 X X XXXXX  "
600 DATA"XXX XXX XX   X     X","XX  X X 
X XXXX XXXXX"
610 DATA"    X   X      X    ","XXX XXXX
XXXXXX XXXXX"
620 DATA"XXX XXX XXXXXXXXXXXX","X     X 
     X     X"
630 DATA"XXX XXXXXXXX  XX X X","  X     
    X  X X  "
640 DATA"X X X XX X XXX X X X","X X X X 
 XX X  X X X"
650 DATA"X  XX XXX    X    XX","XX X   X
X XX XXXX   "
660 DATA"XX   X     X     X X","X XX  XX
X XX X XXX X"
670 DATA"     X   X   X     X","XXX XXX 
XXXXXX XXXXX"
680 DIML$(7,11):POKE53280,0:POKE53281,1
690 FORI=0TO7:FORR=0TO11:READL$(I,R)
700 NEXTR,I:PRINT"♥"
710 GOSUB800
720 A=Z
730 GOSUB800
740 B=Z:IFA=BTHEN 730
750 GOSUB800
760 C=Z:IFA=C OR B=C THEN750
770 GOSUB800
780 D=Z:IFA=D OR B=D OR C=D THEN 770
790 GOTO 810
800 Z=INT(8*RND(0)):RETURN
810 GOSUB820:A=C:B=D
820 FORI=0TO11:PRINTL$(A,I);L$(B,I);:NEX
TI:IFA<>C THEN RETURN
830 PRINT"S
     Z    ";:POKE2023,160:POKE56295,14
```

```
840 FORI=1024TO2023:IFPEEK(I)=24THENPOKE
I,160
850 NEXTI
860 FORI=55296TO56295:POKEI,14:NEXT
870 POKE1027,81
880 H(0)=-40:H(1)=40:H(2)=-1:H(3)=1
890 POKE56332,224:I=1027:TI$="000000":T=
0
900 A=127-PEEK(56320):IFA=0THEN900
905 T=T+1
910 Z=LOG(A)/LOG(2)
920 I=I+H(Z)
930 IF(I-1024)/40=INT((I-1024)/40)ANDH(Z
)=+1THENI=I-H(Z)
940 IF(I-1023)/40=INT((I-1023)/40)AND H(
Z)=-1THENI=I-H(Z)
950 IFI<1027ORI>2023THENI=I-H(Z)
960 IFPEEK(I)=160THENI=I-H(Z)
970 IFPEEK(I)=81THEN900
980 POKEI-H(Z),32:POKEI,81
990 IFI=2018THEN1010
1000 GOTO900
1010 PRINT"[control characters]WOW"
1020 PRINT"SIE BENOETIGTEN ";MID$(TI$,3,
2);"MINUTEN UND         "
1030 PRINTRIGHT$(TI$,2);"SEKUNDEN. SIE M
ACHTENDABEI ";T;"ZUEGE"
```

Schauen Sie sich das Programm an. Bis zur Zeile 120 wird eine kurze Spielanleitung gegeben. Sie ist für das Programm ohne jegliche Bedeutung.

Zeilen 200 bis 670:
Hier stehen die Informationen für die 8 Teile. Was schätzen Sie, wieviel verschiedene Irrgärten lassen sich aus diesen8 Teilen fertigen ? Sie werden staunen: 1680.

Zeilen 680 bis 700:
Einlesen der Information in die Variable L$(A,B);

Zeilen 710 bis 800:
Auswahl von 4 verschiedenen Teilen.

Zeilen 810 bis 870:
Bildschirmaufbau. In Zeile 840 werden die X-e auf dem Bildschirm in reverse Spaces umgewandelt, um die Übersicht zu erhöhen.

Zeilen 880 bis 920:
Hier wird der Joystick abgefragt und die neue Position der Spielfigur berechnet.

Zeilen 950 bis 970:
Bewegt sich die Figur auf dem erlaubten Weg? Wenn nicht, dann wird die alte Position wieder hergestellt.

Zeile 980:
Setzen der Figur.

Zeile 990:
Ziel erreicht, wenn ja dann Sprung nach 1010.

So, damit wäre das Spiel erklärt - ich nehme an, daß Sie mit ihm klar kommen.

Wie Sie beim Ablauf des Spiels feststellen können, ist es

recht einfach, das Ziel zu erreichen. Diesen Schwachpunkt wollen wir jetzt korrigieren.

Tippen Sie bitte folgende Zeilen ein und ändern Sie so das Programm. Der Erfolg wird sein, daß man das Labyrinth nicht mehr sieht. Lediglich der bereits zurückgelegte Weg wird sichtbar.

860 FOR I=55296 TO 56295: POKE I, 1: NEXT I

870 POKE 1027, 81: POKE 56290, 0: POKE 55299, 0

960 IF PEEK (I) = 160 AND ((PEEK (I + 54272) AND 15) "ungleich" 0 THEN I = I - H (Z)

980 POKE I - H (Z), 160: POKE I + 54272, 0: POKE I, 81

Die Änderungen setzen die Zeichenfarbe gleich der Bildschirmfarbe und lassen so das Labyrinh verschwinden.

14.5.2 Car

Im folgenden werden Sie ein Programm finden, das unter die große Gruppe der Autorennen fällt.

Die Spielidee

Die Idee zu diesem Spiel ist folgende: Man sieht aus der Vogelperspektive auf eine Staße herab, auf der gerade ein Autorennen stattfindet. Ein Auto fährt deutlich schneller als die anderen, und überholt somit viele andere. Dieses schnelle Auto können wir steuern, unsere Aufgabe ist es, möglichst lange Zeit die anderen Autos zu überholen, ohne sie zu rammen oder die Begrenzung (den Bordstein) zu berühren.

Konzeption

Sämtliche Autos werden als Sprites ausgelegt, das eigene Fahrzeug kann sich lediglich an der unteren Grenze des Bildschirms waagerecht bewegen. Wir haben es also mit einem Spiel zu tun, das durch folgende Merkmale gekennzeichnet ist:

- Beweglicher Hintergrund

- Spielfigur ist nicht frei beweglich.

Durch den beweglichen Hintergrund soll der Eindruck entstehen, man flöge über dem Geschehen mit derselben Geschwindigkeit dahin, wie das Auto unter uns fährt.

Dieser Eindruck wird noch durch die anderen Autos verstärkt, die rückwarts von oben nach unten auf dem Bildschirm bewegt werden.

Das Programm selbst ist hauptsächlich in Basic gehalten, lediglich die Bewegung der Autos und die Abfrage des Joysticks geschehen in Maschinensprache.

Der in Maschinensprache geschriebene Teil ist in Form eines erweiterten Interrupts vom Basic losgelöst, dies garantiert gleichmäßige Geschwindigkeiten der bewegten Objekte.

Verzweigungskriterien

Entscheidend für die Verzweigungen im Programmablauf sind einzig die Kollisionsregister. Jede Kollision, bei der das Sprite 0 beteiligt ist, deutet entweder auf einen Unfall oder auf das überfahren des Bordsteines hin.

Listing "Car"

```
10 REM INTERRUPT-ROUTINE
20 DATA72,138,72,152,72,169,0,141,20,3,1
69,192,141,21,3,169,224,141,2,220
30 DATA173,0,220,201,123,208,6,206,0,208
,76,40,192,201,119,208,3,238,0,208
40 DATA238,3,208,173,3,208,201,85,208,17
,173,21,208,201,7,240,10,9,4,141
50 DATA21,208,169,0,141,5,208,238,5,208,
173,3,208,201,170,208,17,173,21
60 DATA208,201,15,240,10,9,8,141,21,208,
169,0,141,7,208,238,7,208,104,168
70 DATA104,170,169,127,141,2,220,104,76,
49,234,0
80 FORI=0TO110
90 READA:POKE49152+I,A:NEXT
100 PRINT"█":POKE53280,0:POKE53281,0:V=5
3248
110 A$(0)="                        █ █
█"
120 A$(1)="                                   ██"
:Q=1
130 FORI=832TO894:READA:POKEI,A:NEXT
140 FORI=896TO958:READA:POKEI,A:NEXT
150 POKE2040,13:POKE2041,13:POKE2042,13:
POKEV+39,1:POKEV+40,2:POKEV+41,2
160 POKE2043,13:POKEV+42,2
170 POKEV,168:POKEV+1,170:X=168
180 POKEV+2,168:POKEV+3,0:AX=148:BX=168:
CX=188
190 POKEV+30,0:POKEV+31,0
200 IFQ=0THEN220
210 WAIT56320,111:POKE788,0:POKE789,192:
POKEV+3,0:POKEV+21,3
```

```
220 POKEV+21,3
230 IFPEEK(V+30)AND1ORPEEK(V+31)AND1THEN
POKE2040,14:POKEV+21,1:FORI=0TO500:NEXT:
RUN
240 AX=AX+INT(RND(TI)*5)-2:IFAX<120THENA
X=120
250 BX=BX+INT(RND(TI)*5)-2:IFBX<120THENB
X=120
260 CX=CX+INT(RND(TI)*5)-2:IFCX<120THENC
X=120
270 IFAX>216THENAX=216
280 IFBX>216THENBX=216
290 IFCX>216THENCX=216
300 POKEV+2,AX
310 POKEV+4,BX
320 POKEV+6,CX:PRINTA$(Q):Q=ABS(Q-1):GOT
O230
330 DATA0,0,0,0,126,0,0,126,0,0,255,0
340 DATA12,255,48,15,255,240,12,255,48
350 DATA0,255,0,0,255,0,1,231,128,1,195,
128
360 DATA1,195,128,1,195,128,3,195,192
370 DATA3,195,192,115,255,206,115,255,20
6
380 DATA127,255,254,115,255,206,115,255,
206,0,0,0
390 DATA123,20,0,0,24,0,30,44,77,21,126,
3,240,125,48
400 DATA15,205,240,22,155,41,1,205,156
410 DATA0,155,0,1,201,108,1,105,108
420 DATA1,95,28,1,155,158,23,115,192
430 DATA32,242,132,35,239,216,1,95,28
440 DATA32,242,132,68,155,0,27,125,48,0,
0,0
```

Erklärungen zum Programm:

Zeilen 10 bis 90:
Dieser DATA-Block verkörpert das Maschinenspracheprogramm. Durch die Zeilen 80 und 90 wird es im Bereich ab 49152 installiert.

Zeile 100:
Hier wird der Bildschirm vorbereitet.

Zeilen 130 bis 190:
Installierung der Sprites. Sprite 0 (Auto) in Block 13, Sprite 1 (Kollision) in Block 14. Außerdem werden noch die Farben und Startpositionen bestimmt.

Zeile 210:
Hier wird der Interrupt-Vektor verbogen.

Zeile 230:
Abfrage der Kollisionsregister. Bei einem Zusammenstoß wird das Sprite 0 auf den Block 14 umgeschaltet. Nach kurzer Zeit erfolgt ein neuer Start des Programms.

Zeilen 240 bis 320:
Hier wird die Koordinate der zu überholenden Fahrzeuge bestimmt und verändert. Zeile 320 verändert gleichzeitig noch den Bildschirm.

Zeilen 330 bis 440:
DATA-Block für die Sprites.

Das Basic-Programm bietet nichts, was weiter erwähnenswert wäre, werfen wir also einen Blick auf den Maschinenspracheteil.

Assembler-Listing der im Programm "Car" verwendeten Maschinensprache-Routine.

```
.,  c000  48        pha
.,  c001  8a        txa
.,  c002  48        pha
.,  c003  98        tya          Rettet die Register
.,  c004  48        pha          und IRQ - Vektor
.,  c005  a9 00     lda #$00
.,  c007  8d 14 03  sta $0314
.,  c00a  a9 c0     lda #$c0
.,  c00c  8d 15 03  sta $0315    ------------------
.,  c00f  a9 e0     lda #$e0
.,  c011  8d 02 dc  sta $dc02
.,  c014  ad 00 dc  lda $dc00
.,  c017  c9 7b     cmp #$7b      Joystick Abfrage &
.,  c019  d0 06     bne $c021     steuerung des eigenen
.,  c01b  ce 00 d0  dec $d000     Fahrzeuges
.,  c01e  4c 28 c0  jmp $c028
.,  c021  c9 77     cmp #$77
.,  c023  d0 03     bne $c028
.,  c025  ee 00 d0  inc $d000
------------------
.,  c028  ee 03 d0  inc $d003
.,  c02b  ad 03 d0  lda $d003
.,  c02e  c9 55     cmp #$55
.,  c030  d0 11     bne $c043
.,  c032  ad 15 d0  lda $d015
.,  c035  c9 07     cmp #$07
.,  c037  f0 0a     beq $c043
.,  c039  09 04     ora #$04
.,  c03b  8d 15 d0  sta $d015
.,  c03e  a9 00     lda #$00
.,  c040  8d 05 d0  sta $d005
.,  c043  ee 05 d0  inc $d005
.,  c046  ad 03 d0  lda $d003
.,  c049  c9 aa     cmp #$aa     Bewegungsroutine
```

```
.,  c04b  d0 11     bne $c05e  für die anderen
.,  c04d  ad 15 d0  lda $d015  Fahrzeuge
.,  c050  c9 0f     cmp #$0f
.,  c052  f0 0a     beq $c05e
.,  c054  09 08     ora #$08
.,  c056  8d 15 d0  sta $d015
.,  c059  a9 00     lda #$00
.,  c05b  8d 07 d0  sta $d007
.,  c05e  ee 07 d0  inc $d007
.,  c061  68        pla        -----------------
.,  c062  a8        tay
.,  c063  68        pla
.,  c064  aa        tax        Sprung zum normalen
.,  c065  a9 7f     lda #$7f   Interrupt
.,  c067  8d 02 dc  sta $dc02
.,  c06a  68        pla
.,  c06b  4c 31 ea  jmp $ea31
.,  c06e  00        brk        ------ Ende ------
```

Wie Sie an den Kommentaren am Rand des Listings sehen können, gliedert es sich im wesentlichen in 4 Teile.

Teil 1:

Der 6510 Mikroprozessor verfügt über 3 Register, mit denen er Daten manipulieren kann. Damit diese Register nicht durch unsere Aktionen "verseucht" werden, werden sie vor dem eigentlichen Programm erst einmal gerettet. Das erledigt der Befehl pha für uns. Er bringt den Akku auf den Stack (ein besonderer Bereich im RAM) und damit in Sicherheit. Die Befehle txa und tya schreiben das X- bzw. das Y-Register in den Akku. Mit pla werden auch sie gerettet. Als nächstes wird dann der IRQ- oder Interrupt-Vektor erneuert. Die beiden Befehle lda#$00 und sta$0314 bewirken nichts anderes als der Basic-Befehl POKE 788, 0

Teil 2:

Hier wird der Joystick abgefragt und ausgewertet. Die Befehlsfolge lda$dc00, cmp#$7b und bne$c021 würden in Basic IF PEEK (56320) = 0 THEN $c021 lauten.

Dec$d000 bewegt das Sprite 0 nach links, jmp ist ein Sprungbefehl ähnlich dem GOTO

Mit dem zweiten cmp-Befehl wird geprüft, ob der Joystick nach rechts bewegt wurde.

Teil 3:

Im weiteren gehe ich nicht mehr so genau vor, sondern werde lediglich erklären, was passiert, jedoch die Befehle nicht erläutern.

Es wird Sprite 1 nach unten bewegt und, sobald eine gewisse Grenze überschritten ist, auch Sprite 2 und 3 eingeschaltet. Zum Schluß werden also insgesamt 3 Sprites von oben nach unten auf dem Bildschirm bewegt.

Teil 4:

Hier werden die Register wieder vom Stack geholt und verteilt. Als letztes erfolgt dann der Sprung zur normalen Interrupt-Routine.

14.5.3. 3-D-Autorennen

Das Spiel, mit dem wir uns jetzt beschäftigen wollen, ist ein wenig anspruchsvoller, aber damit komme ich Ihnen ja wohl entgegen.

Die Möglichkeiten der 3D-Darstellung auf einem Home-Computer bzw. auf einem Fernseher sind ja sehr begrenzt. Der ausschlaggebende Grund dafür ist der lediglich zweidimensionale Bildschirm. Es bedeutet, daß die dritte Dimension lediglich vorgegaukelt werden kann. Dem menschlichen Auge werden dabei Bilder gezeigt, mit denen das Gehirn angeregt wird, Parallelen zu wirklich dreidimensionalen Gegebenheiten zu bilden. Eine Möglichkeit so etwas zu erreichen, ist die perspektivische Zeichnung. Eine andere ist der geschickte Einsatz von sich überlappenden Formen. In beiden Techniken werden also dem Betrachter Situationen gezeigt, die er aus dem Alltag kennt und mit Erfahrungswerten verbindet. Ein solcher Wert ist z.B. die Erfahrung, daß von zwei Gegenständen, die sich überlappen, derjenige weiter entfernt ist, der durch den anderen teilweise verdeckt wird.

Die Spielidee

Wir wollen nun ein dreidimensionales Autorennen programmieren. Unsere Straße werden wir perspektivisch zeichnen. Sie darf also keine parallelen Bordsteine haben, sondern diese müssen aufeinander zulaufen und sich irgendwo in der Ferne treffen.

Unter diesen Voraussetzungen wird schon deutlich, daß ein schönes Stück Arbeit vor uns liegt, da es uns aus den gegebenen Zeichen nicht gelingen wird, eine perspektivisch verlaufende Straße zu zeichnen. Wir müssen uns die entsprechenden Zeichen also selbst definieren.

Die Straße

Wie bereits gesagt, werden wir die Straße mit Hilfe von selbstdefinierten Zeichen erstellen. Damit wäre das Problem der Form zumindest schon einmal theoretisch gelöst. Leider ist es aber nicht nur die Form einer geraden Straße, die sich als problematisch erweist, vielmehr gibt es noch zwei weitere Probleme, nämlich:

1. Kurven in der Straße darzustellen;

2. Die Geschwindigkeit der Bewegungen zu simulieren.

Kommen wir zuerst zu den Kurven. Um eine gewisse Abwechselung in den Straßenverlauf zu bringen, ist es notwendig, Kurven zu programmieren. Aber gerade dieses löbliche Ansinnen erweist sich auf den ersten Blick als sehr kompliziert. Schuld daran ist die Perspektive. Sie würde ein ziemlich aufwendiges Programm verlangen, wollte man den Verlauf der Straße berechnen. Diesen Aufwand wollen wir aber nicht betreiben, kommen wir doch auf einen nicht so aufwendigen Weg zu unseren Kurven.

Die Lösung des Kurvenproblems ist relativ einfach, erfordert jedoch ein bißchen Handarbeit.

Statt eine einteilige Straße zu definieren, verwenden wir eine zweiteilige. Darin liegt auch schon der ganze Trick. Der erste Teil der Straße, nennen wir ihn Basis, bleibt immer gleich. Anstelle eines zweiten Teils, der Spitze, definieren wir derer drei. Eine Spitze für einen geraden Streckenverlauf und jeweils eine für eine Links- bzw. Rechtskurve. Immer wenn es an der Zeit ist, eine Kurve auf den Bildschirm zu bringen, tauschen wir einfach die Spitze der Straße komplett aus. Dank der enormen Geschwindigkeit der Maschinensprache gelingt uns das in einer nicht durch das Auge wahrzunehmenden Zeit.

Das zweite Problem hat mir da schon ein wenig mehr Kopfzerbrechen bereitet. Es war eine echte Nuß.

Es geht darum, die Geschwindigkeit des Autos zu simulieren, das man mit dem Joystick steuert. Bevor ich hier eine für mich zufriedenstellende Lösung gefunden hatte, gab es erst einmal eine längere Phase des Probierens.

Experimentiert habe ich mit einem als Sprite ausgebildeten Mittelstreifen oder Objekten neben der Straße (Bäume, Büsche oder Zuschauer). Dies alles ergab aber nicht den von mir erwünschten Effekt. Die im Programm verwirklichte Lösung ergab dann ein Besuch in einer Spielothek. Eine lohnende Sache im übrigen, wenn man Anregungen für Programme benötigt. Bei diesem besagten Besuch schaute ich mir im einzelnen die 3D-Autorennen an. Hier fand ich dann auch die Lösung des Problems.

Bei vielen Programmen wird die Geschwindigkeit dadurch simuliert, daß der Bordstein nicht ein, sondern zweifarbig ist. Er ist z.B. in rote und weiße Farbzonen unterteilt. Diese Zonen wandern dann auf den Spieler zu und erzeugen den Eindruck, man würde sich auf der Straße fortbewegen.

Nachdem ich dann auf diese Weise eine Vorstellung gewonnen hatte, WAS ich programmieren wollte, ging das WIE auch relativ locker von der Hand.

Sie erinnern sich doch sicherlich an den Multi-Color-Modus des Zeichensatzes. Hier kann jedes Zeichen vier Farben (3 Farben & Hintergrund) annehmen. Diesen Modus legte ich der Straße zugrunde.

Als ersten Schritt zeichnete ich dann die Straße auf Millimeterpapier. Das hat den Vorteil, daß man beim Umsetzen jeweils ein Kästchen als einen Bildschirmpunkt annehmen kann.

Der zweite Schritt war dann die Einteilung der Skizze in verschiedene Farbzonen. Die erste Farbzone liegt an der Spitze der Straße. Sie hat eine "Tiefe" von einem Bildschirmpunkt. Die zweite, direkt darunter angeordnet, hat eine Tiefe von 2 Punkten, die dritte 3 Punkte, und so weiter bis man an der Basis der Straße angelangt ist. Um eine deutliche Fortbewegung der Zonen zu erreichen, ist es notwendig, drei verschiedene Farbzonen zu verwenden (zwei davon haben allerdings jeweils dieselbe Farbe). Die drei Farbzonen werden durch die drei punktdefinierenden Bitkombinationen 01, 10 und 11 erzeugt (wir erinnern uns: im Multi-Color-Modus wird jeder Bildschirmpunkt durch jeweils 2 Bits erzeugt). Im Verlauf der Straße wiederholen sich diese Farbzonen immer wieder, so daß man letztendlich eine kontinuierliche Bewegung der Farbstreifen am Rand erhält.

Das Programm zur Bewegung der Farbstreifen ist dann nur noch Routine:

- Register 53282 bestimmt die Farbe der Bitkombination 01.

- Register53283 bestimmt die Farbe der Bitkombination 10.

- Das Color-RAM bestimmt die Farbe der Bitkombination 11.

Das Programm hat also nur die Aufgabe, die beiden Register bzw. das Color-RAM mit den entsprechenden Werten zu beschicken.

Die drei verschiedenen Zustände lassen sich der Tabelle entnehmen:

Zustand	**1**	**2**	**3**
Register 53282:	**R**	**W**	**W**
Register 53283:	**W**	**R**	**W**
Color-RAM :	**W**	**W**	**R**

R bedeutet dabei Rot, W Weiß. Sie sehen also an der Tabelle, wie im Laufe der verschiedenen Zustände die Farbe Rot durch die Zonen wandert.

Es ist vollbracht. Für unsere Straße sind, bisher zwar nur auf dem Papier, alle Probleme gelöst.

Kommen wir jetzt zu den restlichen Routinen:

1. Wir wollen ein Auto steuern können. Wie das geht, haben wir schon bei dem ersten Autorennen gesehen. Wir fragen den Joystick ab und erhöhen oder erniedrigen je nach Stellung des Steuerknüppels das Register 53248.

2. Das Spiel wäre zu einfach, könnte man einfach so durch die Straße fahren. Es muß also ein Hindernis vorhanden sein. Das logischste Hindernis ist ein Auto, das überholt werden muß, aber sich nicht so einfach überholen lassen will. Es wird durch ein Sprite dargestellt, das sich kontinuierlich von oben nach unten bewegt und dabei gleichzeitig von einer Seite zur anderen pendelt.

Diesem Auto gilt es, im Spiel auszuweichen. Ein Zusammenstoß führt genau wie die Berührung des Straßenrandes zu dem Verlust der Spielfigur.

3. Wir haben vor, mit viel Mühe verbundene Kurven zu programmieren. Diese Arbeit wäre umsonst, wenn der einzige Effekt die Veränderung des Hintergrundes wäre. Die Kurven müssen sich also noch anders auswirken. In diesem Spiel wird eine Routine dafür sorgen, daß das Auto zum der Kurve entgegenliegenden Straßenrand gedrängt wird. Dieser Effekt macht das Spiel noch ein wenig naturgetreuer, denn achten Sie einmal darauf, was passiert, wenn Sie in einer Kurve das Lenken vergessen.

4. Wir wollen das Spiel noch ein bißchen interessanter gestalten, indem wir einen Hintergrund programmieren, der am oberen Rand des Bildschirms, also in weiter Ferne ist. Je nachdem ob die Straße geradeaus verläuft oder eine Kurve aufweist, wird dieser Hintergrund stehen oder sich nach der entsprechenden Seite bewegen.

Das Programm

Im Gegensatz zu dem ersten Autoprogramm, das ja quasi nur aus einem Teil bestand, werden wir dieses Programm ein wenig besser gestalten. Sämtliche Routinen werden wir als Unterprogramme ausbilden, das Hauptprogramm besteht dann nur noch aus Maschinensprache-Gosubs.

Der Basic-Teil

Der Basic-Teil hat lediglich die Aufgaben, die Maschinenspracheroutinen zu installieren und den Bildschirminhalt das erste Mal zu zeichnen. Ist dies geschehen, so gerät das Programm in eine Endlosschleife, um eine READY-Meldung auf dem Bildschirm zu verhindern. Alle übrigen Schritte werden durch ein Maschinenspracheprogramm übernommen.

Listing "3-D-Autorennen"

```
10 PRINT"♥"
100 DATA 173, 14, 220, 41, 254, 141, 14,
 220, 1077
101 DATA 165, 1, 41, 251, 133, 1, 206, 2
1, 819
102 DATA 192, 206, 24, 192, 173, 255, 22
3, 141, 1406
103 DATA 255, 23, 173, 21, 192, 208, 239
, 206, 1317
104 DATA 22, 192, 206, 25, 192, 173, 22,
 192, 1024
105 DATA 201, 207, 208, 226, 165, 1, 9,
4, 1021
106 DATA 133, 1, 173, 14, 220, 9, 1, 141
, 692
107 DATA 14, 220, 173, 24, 208, 41, 241,
 9, 930
108 DATA 2, 141, 24, 208, 96, 234, 234,
234, 1173
109 DATA 234, 234, 234, 234, 234, 234, 2
34, 234, 1872
110 DATA 234, 234, 234, 234, 234, 0, 0,
0, 1170
111 DATA 0, 0, 0, 0, 173, 14, 220, 41, 4
48
112 DATA 254, 141, 14, 220, 169, 0, 141,
 20, 959
113 DATA 3, 169, 198, 141, 21, 3, 173, 1
4, 722
114 DATA 220, 9, 1, 141, 14, 220, 96, 23
4, 935
115 DATA 234, 234, 238, 85, 192, 173, 85
, 192, 1433
116 DATA 201, 8, 240, 9, 201, 16, 240, 1
7, 932
117 DATA 201, 24, 240, 26, 96, 162, 1, 1
42, 892
```

```
118 DATA 34, 208, 142, 35, 208, 32, 185,
 192, 1036
119 DATA 96, 162, 1, 142, 34, 208, 232,
142, 1017
120 DATA 35, 208, 32, 180, 192, 96, 162,
 0, 905
121 DATA 142, 85, 192, 232, 142, 35, 208
, 232, 1268
122 DATA 142, 34, 208, 96, 169, 9, 76, 1
87, 921
123 DATA 192, 169, 10, 162, 40, 157, 184
, 217, 1131
124 DATA 157, 224, 217, 157, 88, 218, 15
7, 128, 1346
125 DATA 218, 157, 168, 218, 157, 72, 21
9, 157, 1366
126 DATA 240, 216, 157, 24, 217, 157, 64
, 217, 1292
127 DATA 157, 104, 217, 202, 208, 223, 9
6, 234, 1441
128 DATA 234, 234, 238, 86, 192, 173, 86
, 192, 1435
129 DATA 201, 64, 240, 1, 96, 169, 0, 14
1, 912
130 DATA 86, 192, 238, 87, 192, 174, 87,
 192, 1248
131 DATA 189, 0, 195, 141, 255, 192, 32,
 0, 1004
132 DATA 193, 96, 162, 8, 189, 255, 195,
 157, 1255
133 DATA 255, 4, 189, 7, 196, 157, 39, 5
, 852
134 DATA 189, 15, 196, 157, 79, 5, 189,
23, 853
135 DATA 196, 157, 119, 5, 189, 31, 196,
 157, 1050
136 DATA 159, 5, 202, 208, 223, 96, 162,
 8, 1063
```

```
137 DATA 189, 39, 196, 157, 255, 4, 189,
 47, 1076
138 DATA 196, 157, 39, 5, 189, 55, 196,
157, 994
139 DATA 79, 5, 189, 63, 196, 157, 119,
5, 813
140 DATA 189, 71, 196, 157, 159, 5, 202,
 208, 1187
141 DATA 223, 162, 128, 189, 127, 196, 1
57, 255, 1437
142 DATA 9, 202, 208, 247, 96, 162, 8, 1
89, 1121
143 DATA 79, 196, 157, 255, 4, 189, 87,
196, 1163
144 DATA 157, 39, 5, 189, 95, 196, 157,
79, 917
145 DATA 5, 189, 103, 196, 157, 119, 5,
189, 963
146 DATA 111, 196, 157, 159, 5, 202, 208
, 223, 1261
147 DATA 162, 136, 189, 255, 196, 157, 2
55, 9, 1359
148 DATA 202, 208, 247, 96, 234, 234, 23
4, 174, 1629
149 DATA 87, 192, 189, 0, 195, 201, 2, 2
08, 1074
150 DATA 1, 96, 201, 38, 208, 66, 173, 8
6, 869
151 DATA 192, 240, 1, 96, 173, 79, 4, 14
1, 926
152 DATA 88, 192, 173, 119, 4, 141, 89,
192, 998
153 DATA 173, 159, 4, 141, 90, 192, 162,
 39, 960
154 DATA 189, 39, 4, 157, 40, 4, 189, 79
, 701
155 DATA 4, 157, 80, 4, 189, 119, 4, 157
, 714
```

```
156 DATA 120, 4, 202, 208, 235, 173, 88,
 192, 1222
157 DATA 141, 40, 4, 173, 89, 192, 141,
80, 860
158 DATA 4, 173, 90, 192, 141, 120, 4, 9
6, 820
159 DATA 173, 86, 192, 240, 1, 96, 173,
40, 1001
160 DATA 4, 141, 88, 192, 173, 80, 4, 14
1, 823
161 DATA 89, 192, 173, 120, 4, 141, 90,
192, 1001
162 DATA 162, 0, 189, 41, 4, 157, 40, 4,
 597
163 DATA 189, 81, 4, 157, 80, 4, 189, 12
1, 825
164 DATA 4, 157, 120, 4, 232, 224, 39, 2
08, 988
165 DATA 233, 173, 88, 192, 141, 79, 4,
173, 1083
166 DATA 89, 192, 141, 119, 4, 173, 90,
192, 1000
167 DATA 141, 159, 4, 96, 234, 234, 234,
 174, 1276
168 DATA 87, 192, 189, 0, 195, 201, 2, 2
08, 1074
169 DATA 1, 96, 173, 86, 192, 41, 1, 201
, 791
170 DATA 1, 240, 1, 96, 189, 0, 195, 201
, 923
171 DATA 38, 208, 4, 238, 0, 208, 96, 20
6, 998
172 DATA 0, 208, 96, 234, 234, 234, 173,
 21, 1200
173 DATA 208, 201, 3, 240, 20, 169, 3, 1
41, 985
174 DATA 21, 208, 169, 174, 141, 2, 208,
 169, 1092
```

```
175 DATA 112, 141, 3, 208, 169, 128, 141
, 249, 1151
176 DATA 7, 173, 3, 208, 201, 224, 208,
5, 1029
177 DATA 169, 1, 141, 21, 208, 238, 3, 2
08, 989
178 DATA 173, 3, 208, 41, 15, 208, 3, 23
8, 889
179 DATA 249, 7, 173, 86, 192, 41, 1, 24
0, 989
180 DATA 1, 96, 173, 87, 192, 105, 8, 17
0, 832
181 DATA 189, 0, 195, 201, 38, 208, 4, 2
38, 1073
182 DATA 2, 208, 96, 206, 2, 208, 96, 23
4, 1052
183 DATA 234, 234, 173, 30, 208, 162, 0,
 142, 1183
184 DATA 30, 208, 201, 0, 208, 13, 173,
31, 864
185 DATA 208, 142, 31, 208, 41, 1, 201,
1, 833
186 DATA 240, 1, 96, 169, 1, 141, 21, 20
8, 877
187 DATA 169, 0, 141, 91, 192, 169, 205,
 141, 1108
188 DATA 20, 3, 169, 194, 141, 21, 3, 16
9, 720
189 DATA 13, 141, 248, 7, 96, 206, 91, 1
92, 994
190 DATA 240, 3, 76, 49, 234, 169, 174,
141, 1086
191 DATA 0, 208, 169, 135, 141, 248, 7,
169, 1077
192 DATA 0, 141, 30, 208, 141, 31, 208,
32, 791
193 DATA 92, 192, 76, 49, 234, 0, 0, 0,
643
```

```
194 FOR I=0 TO 93
195 FORR=0 TO 7
196 READ A:POKE49152+8*I+R,A:X=X+A:NEXTR
197 READ A:IFX<>A THEN 5000
198 X=0:S=S+1:NEXTI:S=200
199 REM HAUPTPROGRAMM
200 DATA 72, 152, 72, 138, 72, 169, 224,
 141, 1040
201 DATA 2, 220, 173, 0, 220, 41, 4, 208
, 868
202 DATA 6, 206, 0, 208, 76, 33, 198, 17
3, 900
203 DATA 0, 220, 41, 8, 208, 3, 238, 0,
718
204 DATA 208, 169, 255, 141, 2, 220, 32,
 122, 1149
205 DATA 192, 32, 226, 192, 32, 135, 193
, 32, 1034
206 DATA 31, 194, 32, 70, 194, 32, 154,
194, 901
207 DATA 104, 170, 104, 168, 104, 76, 49
, 234, 1009
280 FOR I=0 TO 7
281 FOR R=0 TO 7
282 READA:POKE 50688+8*I+R,A:X=X+A:NEXTR
283 READ A:IFX<>A THEN 5000
284 X=0:S=S+1:NEXTI:S=300
285 SYS49152:REM CHARACTER GENERATOR
300 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
301 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
302 DATA 0, 0, 0, 0, 0, 0, 24, 0, 24
303 DATA 0, 24, 0, 0, 24, 0, 0, 0, 48
304 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
305 DATA 0, 0, 0, 0,.0, 0, 0, 0, 0
306 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
307 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
308 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
309 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
310 DATA 0, 0, 0, 60, 0, 0, 36, 0, 96
```

```
311 DATA 0, 36, 0, 0, 36, 0, 0, 60, 132
312 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
313 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
314 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
315 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
316 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
317 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
318 DATA 126, 0, 0, 102, 0, 0, 102, 0, 3
30
319 DATA 0, 102, 0, 0, 126, 0, 0, 126, 3
54
320 DATA 0, 0, 126, 0, 0, 0, 0, 0, 126
321 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
322 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
323 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
324 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
325 DATA 0, 0, 0, 0, 0, 126, 0, 0, 126
326 DATA 255, 0, 0, 195, 0, 0, 195, 0, 6
45
327 DATA 0, 195, 0, 0, 195, 0, 0, 255, 6
45
328 DATA 0, 0, 255, 0, 0, 126, 0, 0, 381
329 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
330 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
331 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
332 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
333 DATA 0, 0, 255, 0, 1, 255, 128, 1, 6
40
334 DATA 255, 128, 1, 195, 128, 1, 129,
128, 965
335 DATA 1, 129, 128, 1, 129, 128, 1, 12
9, 646
336 DATA 128, 1, 129, 128, 1, 255, 128,
0, 770
337 DATA 255, 0, 0, 0, 0, 0, 0, 0, 255
338 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
339 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
340 DATA 0, 0, 0, 0, 0, 0, 0, 255, 255
```

```
341 DATA 0, 1, 255, 128, 3, 255, 192, 3,
 837
342 DATA 255, 192, 1, 195, 128, 1, 129,
128, 1029
343 DATA 1, 129, 128, 1, 129, 128, 1, 12
9, 646
344 DATA 128, 3, 129, 192, 3, 129, 192,
1, 777
345 DATA 255, 128, 0, 255, 0, 0, 0, 0, 6
38
346 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
347 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
348 DATA 0, 0, 0, 1, 255, 128, 3, 255, 6
42
349 DATA 192, 7, 255, 224, 7, 255, 224,
7, 1171
350 DATA 129, 224, 3, 0, 192, 3, 0, 192,
 743
351 DATA 3, 0, 192, 3, 0, 192, 3, 0, 393
352 DATA 192, 3, 0, 192, 3, 0, 192, 7, 5
89
353 DATA 0, 224, 7, 255, 224, 3, 255, 19
2, 1160
354 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
355 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
356 DATA 1, 255, 128, 3, 255, 192, 7, 25
5, 1096
357 DATA 224, 7, 255, 224, 7, 255, 224,
3, 1199
358 DATA 129, 192, 3, 0, 192, 3, 0, 192,
 711
359 DATA 3, 0, 192, 3, 0, 192, 3, 0, 393
360 DATA 192, 3, 0, 192, 3, 0, 192, 7, 5
89
361 DATA 0, 224, 7, 255, 224, 7, 255, 22
4, 1196
362 DATA 3, 255, 192, 1, 0, 128, 0, 0, 5
79
```

```
363 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
364 FOR I=0 TO 63
365 FOR R=0 TO 7
366 READ A:POKE 8192+8*I+R,A:X=X+A:NEXTR
367 READA:IFX<>ATHEN 5000
368 X=0:S=S+1:NEXTI:S=380
369 REM SPRITES (AUTOS)
380 DATA 0, 0, 0, 24, 0, 0, 4, 0, 28
381 DATA 12, 0, 112, 56, 0, 48, 0, 48, 2
76
382 DATA 0, 0, 32, 1, 0, 0, 14, 0, 47
383 DATA 64, 0, 12, 67, 49, 68, 1, 17, 2
78
384 DATA 96, 1, 48, 24, 3, 13, 192, 24,
401
385 DATA 0, 12, 0, 6, 0, 0, 0, 0, 18
386 DATA 124, 0, 0, 0, 0, 0, 0, 70, 194
387 DATA 48, 1, 192, 0, 0, 0, 0, 0, 241
388 FOR I=0 TO 7
389 FOR R=0 TO 7
390 READ A:POKE832+8*I+R,A:X=X+A:NEXTR
391 READ A:IF X<>ATHEN 5000
392 X=0:S=S+1:NEXTI:S=400
393 REM SPRITE (EXPLOSION)
400 DATA 63, 63, 63, 63, 15, 15, 3, 3, 2
88
401 DATA 192, 192, 240, 240, 252, 252, 2
52, 252, 1872
402 DATA 5, 5, 1, 1, 0, 0, 0, 0, 12
403 DATA 80, 80, 84, 84, 85, 85, 85, 85,
 668
404 DATA 0, 0, 0, 0, 0, 0, 64, 64, 128
405 DATA 168, 84, 85, 85, 21, 21, 5, 5,
474
406 DATA 0, 0, 0, 0, 64, 64, 80, 80, 288
407 DATA 42, 42, 10, 10, 10, 10, 2, 2, 1
28
408 DATA 128, 128, 128, 128, 160, 160, 1
68, 168, 1168
```

```
409 DATA 3, 3, 3, 2, 0, 0, 0, 0, 11
410 DATA 240, 240, 240, 160, 168, 168, 1
70, 170, 1556
411 DATA 252, 252, 252, 252, 63, 63, 15,
 15, 1164
412 DATA 0, 0, 0, 0, 0, 0, 192, 192, 384
413 DATA 21, 21, 5, 5, 5, 5, 3, 3, 68
414 DATA 0, 0, 64, 64, 64, 64, 240, 240,
 736
415 DATA 10, 10, 1, 1, 1, 1, 0, 0, 24
416 DATA 128, 128, 80, 80, 80, 80, 84, 8
4, 744
417 DATA 2, 2, 0, 0, 0, 0, 0, 0, 4
418 DATA 160, 160, 168, 168, 40, 40, 42,
 42, 820
419 DATA 252, 252, 63, 63, 15, 15, 3, 2,
 665
420 DATA 0, 0, 0, 0, 0, 0, 192, 128, 320
421 DATA 5, 5, 5, 5, 1, 3, 0, 0, 24
422 DATA 0, 0, 64, 64, 64, 192, 240, 240
, 864
423 DATA 15, 15, 15, 15, 63, 63, 255, 25
5, 696
424 DATA 252, 252, 240, 240, 192, 192, 1
92, 192, 1752
425 DATA 0, 0, 0, 0, 0, 0, 1, 1, 2
426 DATA 21, 21, 85, 85, 85, 85, 84, 84,
 550
427 DATA 64, 64, 64, 64, 0, 0, 0, 0, 256
428 DATA 0, 0, 0, 0, 1, 1, 5, 5, 12
429 DATA 170, 85, 84, 84, 84, 84, 80, 80
, 751
430 DATA 2, 2, 10, 10, 10, 10, 42, 42, 1
28
431 DATA 168, 168, 160, 160, 160, 160, 1
28, 128, 1232
432 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
433 DATA 15, 15, 63, 42, 42, 42, 170, 17
0, 559
```

```
434 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
435 DATA 240, 240, 192, 128, 0, 0, 0, 0,
 800
436 DATA 0, 0, 0, 0, 0, 0, 3, 3, 6
437 DATA 63, 63, 63, 63, 252, 252, 240,
240, 1236
438 DATA 0, 0, 0, 0, 1, 1, 15, 15, 32
439 DATA 84, 84, 84, 84, 80, 80, 192, 19
2, 880
440 DATA 2, 2, 5, 5, 5, 5, 21, 21, 66
441 DATA 160, 160, 64, 64, 64, 64, 0, 0,
 576
442 DATA 10, 10, 10, 10, 40, 40, 168, 16
8, 456
443 DATA 128, 128, 0, 0, 0, 0, 0, 0, 256
444 DATA 0, 0, 0, 0, 0, 0, 3, 2, 5
445 DATA 63, 63, 60, 60, 240, 240, 192,
128, 1046
446 DATA 0, 0, 0, 0, 1, 3, 15, 15, 34
447 DATA 80, 80, 80, 80, 64, 192, 0, 0,
576
448 DATA 2, 2, 2, 2, 0, 0, 0, 0, 8
449 DATA 128, 128, 160, 160, 80, 80, 20,
 20, 776
450 DATA 240, 240, 60, 60, 8, 8, 10, 10,
 636
451 DATA 4, 4, 4, 4, 1, 3, 0, 0, 20
452 DATA 0, 0, 0, 0, 0, 0, 192, 192, 384
453 DATA 3, 3, 2, 2, 0, 0, 0, 0, 10
454 DATA 0, 0, 0, 0, 128, 128, 32, 16, 3
04
455 DATA 192, 128, 128, 64, 16, 16, 12,
12, 568
456 DATA 2, 2, 2, 2, 5, 5, 20, 20, 58
457 DATA 128, 128, 128, 128, 0, 0, 0, 0,
 512
458 DATA 15, 15, 60, 60, 40, 40, 160, 16
0, 550
459 DATA 0, 0, 0, 0, 0, 0, 3, 3, 6
```

```
460 DATA 16, 16, 64, 64, 64, 192, 0, 0,
416
461 DATA 0, 0, 2, 2, 2, 2, 8, 4, 20
462 DATA 192, 192, 128, 128, 0, 0, 0, 0,
 640
463 DATA 3, 2, 8, 4, 4, 4, 48, 48, 121
470 FOR I=0 TO 63
471 FOR R=0 TO 7
472 READ A:POKE 2048+8*I+R,A:X=X+A:NEXTR
473 READ A:IFX<>A THEN 5000
474 X=0:S=S+1:NEXTI:S=500
475 REM STRASSE (BITMUSTER)
500 DATA 0, 0, 0, 15, 63, 63, 255, 255,
651
501 DATA 0, 0, 0, 3, 255, 255, 255, 255,
 1023
502 DATA 0, 3, 15, 255, 255, 255, 255, 2
55, 1293
503 DATA 0, 240, 240, 252, 255, 255, 255
, 255, 1752
504 DATA 0, 0, 0, 0, 0, 252, 255, 255, 7
62
505 DATA 0, 3, 15, 15, 63, 255, 255, 255
, 861
506 DATA 192, 240, 240, 240, 252, 255, 2
55, 255, 1929
507 DATA 0, 0, 0, 0, 0, 0, 192, 240, 432
508 DATA 0, 0, 0, 0, 0, 3, 15, 63, 81
509 DATA 0, 0, 0, 0, 0, 0, 60, 255, 315
520 FOR I=0 TO 9
521 FOR R=0 TO 7
522 READ A:POKE 2696+8*I+R,A:X=X+A:NEXTR
523 READ A:IFX<>A THEN 5000
524 X=0:S=S+1:NEXTI:S=600
525 REM HINTERGRUND (BITMUSTER)
600 DATA 2, 2, 0, 0, 0, 0, 0, 0, 4
601 DATA 128, 128, 160, 160, 20, 20, 5,
5, 626
602 DATA 15, 15, 3, 3, 0, 0, 0, 0, 36
```

```
603 DATA 192, 192, 240, 240, 40, 40, 10,
 10, 964
604 DATA 20, 20, 5, 5, 0, 0, 0, 0, 50
605 DATA 0, 0, 64, 64, 80, 240, 60, 60,
568
606 DATA 15, 15, 0, 0, 0, 0, 0, 0, 30
607 DATA 0, 0, 160, 160, 10, 10, 0, 0, 3
40
608 DATA 0, 0, 0, 0, 0, 0, 160, 160, 320
609 DATA 0, 0, 128, 64, 5, 5, 0, 0, 202
610 DATA 0, 0, 0, 0, 0, 0, 240, 240, 480
611 DATA 8, 8, 40, 40, 16, 16, 16, 16, 1
60
612 DATA 3, 3, 3, 3, 2, 2, 2, 2, 20
613 DATA 1, 1, 1, 1, 1, 3, 3, 3, 14
614 DATA 12, 12, 8, 8, 2, 2, 2, 1, 47
615 DATA 255, 170, 32, 16, 16, 16, 48, 4
8, 601
616 DATA 160, 160, 40, 40, 20, 20, 4, 4,
 448
617 DATA 3, 3, 3, 3, 0, 0, 0, 0, 12
618 DATA 0, 0, 0, 0, 128, 128, 160, 160,
 576
619 DATA 1, 1, 1, 1, 1, 3, 3, 3, 14
620 DATA 0, 0, 0, 0, 0, 0, 2, 1, 3
621 DATA 192, 192, 128, 128, 128, 128, 0
, 0, 896
622 DATA 255, 170, 8, 4, 16, 16, 48, 48,
 565
623 DATA 2, 2, 8, 8, 20, 20, 16, 16, 92
624 DATA 128, 128, 0, 0, 0, 0, 0, 0, 256
625 DATA 3, 3, 15, 15, 40, 40, 160, 160,
 436
626 DATA 192, 192, 0, 0, 0, 0, 0, 0, 384
627 DATA 0, 0, 1, 1, 5, 15, 240, 240, 50
2
628 DATA 80, 80, 64, 64, 0, 0, 0, 0, 288
629 DATA 0, 0, 0, 0, 2, 2, 10, 5, 19
```

```
630 DATA 3, 3, 40, 40, 128, 128, 0, 0, 3
42
631 DATA 0, 0, 0, 0, 5, 5, 240, 240, 490
632 DATA 3, 2, 168, 84, 0, 0, 0, 0, 257
640 FOR I=0 TO 32
641 FOR R=0 TO 7
642 READ A:POKE50304+8*I+R,A:X=X+A:NEXTR
643 READ A:IF X<>A THEN5000
644 X=0:S=S+1:NEXTI:S=700
645 REM KURVEN (BITMUSTER)
700 DATA 32, 32, 32, 63, 55, 32, 32, 32,
 310
701 DATA 32, 32, 61, 62, 53, 54, 32, 32,
 358
702 DATA 32, 59, 60, 32, 32, 51, 52, 32,
 350
703 DATA 32, 58, 32, 32, 32, 32, 50, 32,
 300
704 DATA 56, 57, 32, 32, 32, 32, 48, 49,
 338
705 DATA 79, 73, 74, 32, 32, 32, 32, 32,
 386
706 DATA 78, 32, 70, 71, 72, 32, 32, 32,
 419
707 DATA 77, 32, 32, 32, 68, 69, 32, 32,
 374
708 DATA 76, 32, 32, 32, 32, 66, 67, 32,
 369
709 DATA 75, 32, 32, 32, 32, 32, 64, 65,
 364
710 DATA 32, 32, 32, 32, 32, 79, 80, 70,
 389
711 DATA 32, 32, 32, 77, 78, 32, 68, 69,
 420
712 DATA 32, 32, 75; 76, 32, 32, 67, 32,
 378
713 DATA 32, 73, 74, 32, 32, 32, 65, 66,
 406
```

```
714 DATA 71, 72, 32, 32, 32, 32, 32, 64,
 367
720 FOR I=0 TO 14
721 FOR R=0 TO 7
722 READ A:POKE50176+8*I+R,A:X=X+A:NEXTR
723 READ A:IF X<>A THEN5000
724 X=0:S=S+1:NEXTI:S=800
725 REM KURVEN (ZEICHEN)
800 DATA 38, 38, 38, 2, 2, 2, 2, 38, 160
801 DATA 38, 85, 85, 85, 38, 38, 38, 38,
 445
802 DATA 85, 85, 85, 38, 2, 2, 2, 2, 301
803 DATA 2, 2, 2, 2, 38, 38, 38, 38, 160
804 DATA 38, 38, 38, 38, 38, 2, 2, 2, 19
6
805 DATA 85, 85, 85, 85, 2, 38, 2, 2, 38
4
806 DATA 38, 2, 85, 2, 38, 38, 2, 38, 24
3
807 DATA 2, 85, 85, 2, 38, 38, 38, 38, 3
26
808 DATA 2, 2, 2, 2, 2, 2, 38, 38, 88
809 DATA 2, 2, 2, 2, 2, 2, 2, 2, 16
810 DATA 38, 38, 38, 38, 38, 38, 38, 38,
 304
811 DATA 38, 2, 85, 2, 38, 38, 2, 85, 29
0
812 DATA 85, 85, 2, 38, 38, 38, 2, 2, 29
0
813 DATA 2, 2, 38, 2, 85, 85, 85, 2, 301
814 DATA 38, 38, 38, 38, 38, 2, 85, 85,
362
815 DATA 85, 85, 85, 85, 85, 85, 85, 85,
 680
816 DATA 2, 38, 38, 38, 38, 38, 38, 38,
268
817 DATA 2, 2, 38, 2, 85, 2, 38, 2, 171
818 DATA 2, 2, 2, 2, 2, 2, 2, 38, 52
```

```
819 DATA 38, 2, 85, 85, 85, 2, 38, 38, 3
73
820 DATA 38, 38, 38, 38, 38, 38, 2, 85,
315
821 DATA 85, 85, 85, 85, 85, 85, 85, 85,
 680
822 DATA 2, 38, 38, 38, 38, 2, 85, 2, 24
3
823 DATA 38, 2, 2, 38, 2, 85, 2, 38, 207
824 DATA 2, 2, 38, 2, 38, 2, 85, 2, 171
825 DATA 38, 2, 2, 2, 38, 38, 38, 38, 19
6
826 DATA 2, 2, 2, 2, 2, 38, 2, 85, 135
827 DATA 2, 85, 85, 2, 38, 2, 2, 2, 218
828 DATA 38, 38, 38, 2, 2, 2, 2, 2, 124
829 DATA 38, 38, 38, 38, 38, 2, 85, 85,
362
830 DATA 85, 85, 85, 2, 2, 2, 2, 38, 301
831 DATA 38, 38, 2, 2, 2, 2, 38, 38, 160
840 FOR I=0 TO 31
841 FOR R=0 TO 7
842 READ A:POKE 49920+8*I+R,A:X=X+A:NEXT
R
843 READA:IF X<>A THEN5000
844 X=0:S=S+1:NEXTI
845 REM STRASSENVERLAUF
1000 REM STRASSE ZEICHNEN
1010 POKE53270,PEEK(53270)OR16
1020 FORI=55296TO56295:POKEI,PEEK(I)OR8
1030 NEXT
1040 V=53248:POKEV+21,1:POKEV+1,200:POKE
2040,32:POKEV,174
1050 POKEV+33,7:POKEV+34,2:POKEV+35,1:PO
KEV+36,1
1060 PRINT"                    ";
1070 PRINT"                                  
      ";
```

```
1080 PRINT"[illegible]
                "; 
1090 PRINT"[illegible]
                 "; 
1100 PRINT"[illegible]
                 "; 
1110 PRINT"                   ?7       "
1120 PRINT"                  =>56      "
1130 PRINT"                 ;<  34     "
1140 PRINT"                 :    2     "
1150 PRINT"                89    01    "
1160 PRINT"               ./      UV   "
1170 PRINT"              ,-        ST  "
1180 PRINT"              *+        QR"
1190 PRINT"             ()          OP"
1200 PRINT"            &'            MN"
1210 PRINT"           $%              KL
"
1220 PRINT"           !#              IJ
"
1230 PRINT"          ↑←                G
H"
1240 PRINT"         £]
EF"
1250 PRINT"        YZ[
BCD"
1260 PRINT"        WX
 @A"
1270 SYS49244
1280 GOTO1280
5000 PRINT"DATA ERROR IN ZEILE";S;"!!!!!
"
5010 END
```

Wie wir sehen, besteht das Programm fast nur aus DATA-Statements.

Es gibt insgesamt 9 DATA-Blocks, die Aufgabe jedes einzelnen Blockes wird durch eine REM-Zeile am Ende des entsprechenden Laders verdeutlicht:

```
Block  1: Maschinenspracheprogramm

Block  2: Hauptprogramm (Maschinensprache)

Block  3: Verlegeroutine für den Character Generator

Block  4: Sprite-DATAs (Autos)

Block  5: Sprite-DATAs (Explosion)

Block  6: Bitmuster für die selbstdefinierten Zeichen
          (Straße)

Block  7: Bitmuster für den Hintergrund

Block  8: Bitmuster für die Kurven

Block  9: Zeichen für Kurven

Block 10: Straßenverlauf
```

Jede DATA-Zeile hat eine Länge von 9 DATA-Statements, dabei ist das 9. Element eine Prüfsumme für die entsprechende Zeile. Tippen Sie das Programm am besten unter Beibehaltung der Zeilennummern ein, mit dem Eroflg, daß das Programm bei einem Tippfehler in den DATA-Zeilen Ihnen die fehlerhafte Programmzeile ausgibt.

Sind die DATAs eingelesen, so werden vom Programm nur noch der Multi-Color-Modus eingeschaltet (Zeilen 1010 bis 1030), Sprites eingeschaltet (1040 & 1050), die Straße gezeichnet

(bis 1260) und das Maschinenspracheprogramm aufgerufen (1270). Zeile 1280 ist die besagte Endlosschleife, die Zeilen 5000 und 5010 dienen dazu, um fehlerhafte DATA Zeilen auszugeben.
Bevor Sie das Programm starten, sollte es erstens abgespeichert sein, zweitens müssen Sie einen Bereich des RAMs geschützt haben.

Wir verlegen den Basic-Start auf die Adresse 10240. Dies geschieht durch:

POKE 44, 40: POKE 10240, 0: NEW

Der Maschinensprache-Teil

Der erste Schritt, der getan wird, ist das Verlegen des Character-Generators. Hier wird der Generator ausgelesen und in den Bereich ab 2048 kopiert. Diese Routine kann auch immer dann verwendet werden, wenn Sie bei anderen Programmen den Zeichensatz verschieben wollen.

Zu beachten ist, daß sich die Routine selbständig verändert. Wird sie ein zweites Mal angesprungen, so kann das zum Aufhängen des Rechners führen.

```
        lda $dc0e         Interrupt ausschalten
        and #$fe
        sta $dc0e
        lda $01           ROM selektieren
        and #$fb
        sta $01
Loop1   dec $ROMl         Verschieberoutine
        dec $RAMl
        lda $ROMlROMh
        sta $RAMlRAMh
        lda $ROMl
        bne $Loop1
        dec $ROMh
        dec $RAMh
        lda $ROMh
        cmp #$cf
        bne $Loop1
        lda $01           ROM ausschalten (Normalzustand)
        ora #$04
        sta $01
        lda $dc0e         Interrupt einschalten
        ora #$01
        sta $dc0e
        lda $d018         Character Generator verlegen
        and #$f1
```

```
ora $02
sta $d018
rts                 Rücksprung
```

Zur Verdeutlichung, was im einzelnen abläuft, möchte ich hier ein Basic-Programm einfügen, das dieselben Aufgaben erfüllt.

```
10 POKE 56334, PEEK (56334) AND 254: Schaltet den Interrupt
ab.
20 POKE 1, PEEK (1) AND 251: ROM selektieren
30 FOR I = 0 TO 4095: Verschieberoutine
40 POKE 2048 + I, PEEK (53248 + I)
50 NEXT I
60 POKE 1, PEEK (1) OR 4: ROM in den Orginalzustand bringen.
70 POKE 56334, PEEK (56334) OR 1: Interrupt einschalten.
80 POKE 53272, (PEEK (53272) AND 241) OR 2: Generator
verschieben
90 RETURN
```

Für dieses Programm finden Sie im Teil 4 dieses Buches einen Basic-Lader, so daß Sie beim Verwenden dieses Programms keine Probleme haben dürften. Eins muß noch gesagt werden, das Programm ist so, wie es hier steht, nur an der vorgesehenen Stelle verwendbar, da es sich selbst verändert.

Die Unterprogramme des Hauptprogramms

1. Unterprogramm: Joystickabfrage

Das erste Unterprogramm, das angesprungen wird, sorgt für die Abfrage des Joysticks und die Steuerung des Sprites (Spielfigur).

```
        lda $dc00         Auslesen des Joystickports
        cmp #$7b
        bne $Rechts
        dec $d000         Bewegt Sprite eine Position nach
                          links
Rechts  cmp #$77
        bne $Ende
        inc $d000 Bewegt Sprite eine Position nach
rechts
Ende    rts               Rücksprung ins Hauptprogramm
```

In Basic sähe das Programm folgendermaßen aus:

```
10 A = PEEK (56320): Auslesen des Joystickports
20 IF A = 123 THEN POKE 53248, PEEK (53248) - 1 : GOTO 40
30 IF A = 119 THEN POKE 53248, PEEK (53248) + 1
40 RETURN
```

2. Unterprogramm: Farbänderung

Die Bewegung der Straße geschieht durch einen kontinuierlichen Farbwechsel der Bordsteine. Für diesen Wechsel ist diese Routine verantwortlich:

```
         inc $c055        Zähler für die Änderung
         lda $c055
         cmp #$08
         beq $Zustand 1
         cmp #$10
         beq $Zustand 2
         cmp #$18
         beq $Zustand 3
         rts              Rücksprung ins Hauptprogramm
Zustand1 ldx #$01         Farbe Weiß
         stx $d022        Register 53282
         stx $d023        Register 53283
```

```
        jsr $Color 2
        rts                 Rücksprung ins Hauptprogramm
Zustand2 ldx #$01           Farbe Weiß
        stx $d022           Register 53282
        inx                 Farbe Rot
        stx $d023           Register 53283
        jsr $Color 1
        rts                 Rücksprung ins Hauptprogramm
Zustand3 ldx #$00
        stx $c055           Zähler auf Null setzen
        inx                 Farbe Weiß
        stx $d023           Register 53283
        inx                 Farbe Rot
        stx $d022           Register 53282
        rts                 Rücksprung ins Hauptprogramm
Color 1  lda #$09           Farbe Weiß & Multi Color Bit
        jmp $Anfang
Color 2  lda #$0a           Farbe Rot & Multi Color Bit
Anfang   ldx #$28           Schleifenzähler
Loop 2   sta $d9b8,x        Positinen im Color RAM
        sta $d9e0,x
        sta $da58,x
        sta $da80,x
        sta $daa8,x
        sta $db48,x
        sta $db48,x
        sta $d8f0,x
        sta $d918,x
        sta $d940,x
        sta $d968,x
        dex
        bne $Loop 2
        rts                 Rücksprung ins Unterprogramm
                            Farbänderung
```

Ein entsprechendes Basic-Programm sähe so aus:

```
10 A = A + 1
20 IF A = 8 THEN GOTO 100
30 IF A = 16 THEN GOTO 200
40 IF A = 24 THEN GOTO 300
50 RETURN 100 POKE 53282, 1
110 POKE 53283, 1
120 GOSUB 420
130 RETURN 200 POKE 53282, 1
210 POKE 53283, 2
220 GOSUB 400
230 RETURN
300 A = 0
310 POKE 53283, 1
320 POKE 53282, 2
330 RETURN
400 X = 9
410 GOTO 430
420 X = 10
430 FOR I = 0 TO 39
440 POKE 55536 + I, X
450 POKE 55576 + I, X
460 POKE 55616 + I, X
470 POKE 55656 + I, X
480 POKE 55736 + I·, X
490 POKE 55776 + I, X
500 POKE 55896 + I, X
510 POKE 55936 + I, X
520 POKE 55976 + I, X
530 POKE 56136 + I, X
540 NEXT I
550 RETURN
```

Dieses Basic-Programm mag Ihnen ein wenig umständlich erscheinen. Ich gebe Ihnen da recht. Der Grund, warum es in dieser Form steht, ist der, daß es relativ genau beschreibt, wie das Maschinenspracheprogramm abläuft. Es erfüllt also seinen Zweck, in dem es Ihnen ermöglicht, das Maschinenspracheprogramm zu verstehen.

3. Unterprogramm: Veränderung des Straßenverlaufs

Diese Routine zeichnet die Kurven der Straße. Dabei wird der Straßenverlauf bei jedem 64. Interrupt verändert. Die neue Form, Kurve oder gerades Stück, wird dabei aus einer Tabelle im Bereich $C300-$C3FF entnommen. Dies ist jedoch nicht die einzige Tabelle, aus der das Programm Informationen bezieht.

Weitere Tabellen sind:

```
$C400-$C427: Zeichen für Straßenverlauf (gerades Stück)

$C428-$C44F: Zeichen für Straßenverlauf (Linkskurve)

$C450-$C477: Zeichen für Straßenverlauf (Rechtskurve)

$C480-$C4FF: Bitmuster für Linkskurve

$C500-$C587: Bitmuster für Rechtskurve
```

```
         inc $c056       Zähler für den Formwechsel
         lda $c056
         cmp #$40
         beq $Änder
         rts             Rücksprung ins Hauptprogramm
Änder    lda #$00
         sta $c056       Zähler Straße Initialisieren
         inc $c057       Zeiger Straßenverlauf
         ldx $c057
         lda $c300,X     Holt die Form der nächsten Straße
         sta $Sprung     Ändert das Low-Byte des Sprunges
         jsr $Sprung
         rts             Rücksprung ins Hauptprogramm
Sprung1  ldx #$08        1. mögliche Sprungadresse
Loop 1   lda $c3ff,x     Liest  Zeichen  aus der Tabelle
         sta $04ff,x     Straßenform, und zeichnet diese
         lda $c407,x     auf den Bildschirm. Dieser Teil
```

```
         sta $0527,x     zeichnet die gerade Straße
         lda $c40f,x
         sta $054f,x
         lda $c417,x
         sta $0577,x
         lda $c41f,x
         sta $059f,x
         dex
         bne $Loop 1
         rts             Rücksprung zum Unterprogramm
Sprung2  ldx #$08        2. mögliche Sprungadresse
Loop 2   lda $c427,x     Siehe Sprung 1: Zeichnen der
         sta $04ff,x     Linkskurve
         lda $c42f,x
         sta $0527,x
         lda $c437,x
         sta $054f,x
         lda $c43f,x
         sta $0577,x
         lda $c447,x
         sta $059f,x
         dex
         bne $Loop 2
         ldx #$80
Loop 3   lda $c47f,x     Definiert die entsprechenden
         sta $09ff,x     Zeichen für die Linkskurve um
         dex
         bne $Loop 3
         rts             Rücksprung ins Unterprogramm
Sprung3  ldx #$08        3. mögliche Sprungadresse
Loop 4   lda $c44f,x     Siehe Sprung 1: Zeichnen der
         sta $04ff,x     Rechtskurve
         lda $c457,x
         sta $0527,x
         lda $c45f,x
         sta $054f,x
         lda $c467,x
         sta $0577,x
         lda $c46f,x
```

```
        sta $059f,x
        dex
        bne $Loop 4
        ldx #$88
Loop 5  lda $c4ff,x     Definiert die entsprechenden
        sta $09ff,x     Zeichen für die Rechtskurve um
        dex
        bne $Loop 5
        rts             Rücksprung ins Unterprogramm
```

Hier, wie schon gewohnt, ein entsprechendes Basic-Programm:

```
10 A = A + 1
20 IF A = 64 THEN GOTO 40
30 RETURN
40 A = 0
50 READ B
60 ON B GOSUB 100, 200, 400
70 RETURN
100 FOR I = 0 TO 7
110 POKE 1272 + I, PEEK (50176 + I)
120 POKE 1312 + I, PEEK (50184 + I)
130 POKE 1352 + I, PEEK (50192 + I)
140 POKE 1392 + I, PEEK (50200 + I)
150 POKE 1432 + I, PEEK (50208 + I)
160 NEXT I
170 RETURN
200 FOR I = 0 TO 7
210 POKE 1272 + I, PEEK (50216 + I)
220 POKE 1312 + I, PEEK (50224 + I)
230 POKE 1352 + I, PEEK (50232 + I)
240 POKE 1392 + I, PEEK (50240 + I)
250 POKE 1432 + I, PEEK (50248 + I)
260 NEXT I
270 FOR I = 0 TO 79
280 POKE 2560 + I, PEEK (50304 + I)
290 NEXT I
```

```
300 RETURN
400 FOR I = 0 TO 7
410 POKE 1272 + I, PEEK (50256 + I)
420 POKE 1312 + I, PEEK (50264 + I)
430 POKE 1352 + I, PEEK (50272 + I)
440 POKE 1392 + I, PEEK (50280 + I)
450 POKE 1432 + I, PEEK (50288 + I)
460 NEXT I
470 FOR I = 0 TO 87
480 POKE 2560 + I, PEEK (50432 + I)
490 NEXT I
500 RETURN
```

Zu den Zeilen 50 und 60 im Basic-Programm muß gesagt werden, daß sie nur sehr unzureichend das Maschinenspracheprogramm simulieren können.

Der Befehl READ zeigt zwar bestimmte Parallelen zu dem was passiert, dasselbe bewirkt er aber nicht.

Im Maschinenspracheprogramm wird hier ein Zeichen aus der Tabelle Straßenverlauf geladen und der Zähler um eins erhöht. Erreicht er den Wert 256, so wird praktisch ein RESTORE ausgeführt. Der Zeiger geht wieder auf den Wert 0. Im Basic passiert dies nicht.

Die Zeile 60 versucht, die Maschinenprogrammstelle sta $Sprung: jsr $Sprung zu simulieren. Richtiger müßte es heißen: GOSUB B. Diesen Befehl erlaubt unser Basic allerdings nicht.

4. Unterprogramm: Verschiebung des Hintergrundes

Hier wird der Hintergrund am oberen Bildschirmrand bewegt. Je nach Art der Strecke wird der Hintergrund nach links oder rechts bewegt. Dabei werden die Zeichen, die am linken Rand überflüssig werden, am rechten Rand wieder angefügt und umgekehrt.

```
         ldx $c057        Zeiger Straßenverlauf
         lda $c300,x      Aktueller Straßenverlauf
         cmp #$05
         bne $Vergl.
         rts              Rücksprung ins Hauptprogramm
Vergl.   cmp #$29
         bne $Rechts
         lda $c056        Zähler Straßenverlauf
         beq $Links
         rts              Rücksprung ins Hauptprogramm
Links    lda $044f        Retten des rechten Bildschirm-
         sta $c058        randes
         lda $0477
         sta $c059
         lda $049f
         sta $c05a
         ldx #$27
Loop 1   lda $0427,x      Verschieben des Hintergrundes
         sta $0428,x
         lda $044f,x
         sta $0450,x
         lda $0477,x
         sta $0478,x
         dex
         bne $Loop 1
         lda $c058        Setzen der geretteten Zeichen an
         sta $0428        den linken Bildschirmrand
         lda $c059
         sta $0450
```

```
        lda $c05a
        sta $0478
        rts              Rücksprung ins Hauptprogramm
Rechts  lda $c056        Zähler Straßenverlauf
        beq $Retten
        rts              Rücksprung ins Hauptprogramm
Retten  lda $0428        Retten des linken Bildschirm-
        sta $c058        randes
        lda $0450
        sta $c059
        lda $0478
        sta $c05a
        ldx #$00
Loop 2  lda $0429,x      Verschieben des Bildschirms
        sta $0428,x
        lda $0451,x
        sta $0450,x
        lda $0479,x
        sta $0478,x
        inx
        cpx #$27
        bne $Loop 2
        lda $c058        Setzen der geretteten Zeichen an
        sta $044f        den rechten Bildschirmrand
        lda $c059
        sta $0477
        lda $c05a
        sta $049f
        rts              Rücksprung ins Hauptprogramm
```

Das entsprechende Basic-Programm:

```
10 A = PEEK (49920 + PEEK (49239))
20 IF A = 5 THEN RETURN
30 IF A = 41 THEN GOTO 200
40 IF PEEK (49238) = 0 THEN RETURN
50 POKE 49240, PEEK (1103)
```

```
60 POKE 49241, PEEK (1143)
70 POKE 49242, PEEK (1183)
80 FOR I = 39 TO 0 STEP -1
90 POKE 1063 + I, PEEK (1064 + I)
100 POKE 1103 + I, PEEK (1104 + I)
110 POKE 1143 + I, PEEK (1144 + I)
120 NEXT I
130 POKE 1064, PEEK (49240)
140 POKE 1104, PEEK (49241)
150 POKE 1144, PEEK (49242)
160 RETURN
200 IF PEEK (49238) = 0 THEN RETURN
210 POKE 49240, PEEK (1064)
220 POKE 49241, PEEK (1104)
230 POKE 49242, PEEK (1144)
240 FOR I = 0 TO 39
250 POKE 1064 + I, PEEK (1063 + I)
260 POKE 1104 + I, PEEK (1103 + I)
270 POKE 1144 + I, PEEK (1143 + I)
280 NEXT I
290 POKE 1103, PEEK (49240)
300 POKE 1143, PEEK (49241)
310 POKE 1183, PEEK (49242)
320 RETURN
```

5. Unterprogramm: Fliehkraft

Hier steht die Routine, die das Auto an den Straßenrand "drängt", wenn eine Kurve auf dem Bildschirm erscheint.

```
ldx $c057        Zeiger Straßenverlauf
lda $c300,x
cmp #$05         Geradeaus?
bne $Kurve
rts              Rücksprung ins Hauptprogramm
```

```
Kurve    lda $c056         Zähler Straßenverlauf
         and #$01 Bit 0 gesetzt? Diese Routine läuft
         cmp #$01          nur jeden 2. Interrupt
         beq $Vergl.
         rts               Rücksprung ins Hauptprogramm
Vergl.   lda $c300,x
         cmp #$29          Linkskurve?
         bne $Rects
         inc $d000         Versetzt das Auto nach rechts
         rts               Rücksprung ins Hauptprogramm
Rechts   dec $d000         Versetzt das Auto nach links
         rts               Rücksprung ins Hauptprogramm
```

Das entsprechende Basic-Programm:

```
10 IF PEEK (49920 + PEEK (49239)) = 5 THEN RETURN
20 IF PEEK (49238) AND 1 "ungleich" 0 THEN RETURN
30 IF PEEK (49920 + PEEK (49239)) = 41 THEN GOTO 100
40 POKE 53248, PEEK (53248) + 1
50 RETURN
100 POKE 53248, PEEK (53248) - 1
110 RETURN
```

6. Unterprogramm: Das Hindernis

Kommmen wir zu der Routine, die uns das Leben schwer machen soll. Hier wird ein Fahrzeug programmiert, das uns entgegenkommen wird. Es sind zwei Punkte zu beachten. Erstens, daß das Fahrzeug in der Zeit, die es benötigt, um vom oberen Rand zum unteren Rand zu gelangen, ständig an Größe gewinnt. Zweitens, daß das Fahrzeug nicht in gerader Linie herunterkommt, sondern hin und her pendelt.

1. Problem: Im Programm wird bei jedem 16. Durchgang das Bild des Sprites ausgewechselt, und zwar wird jedesmal ein Fahrzeug mit größerem Format als Sprite genommen. So wird die Größe in acht Schritten von Punktgröße bis zur entgültigen Größe fast kontinuierlich verändert.

2. Problem: Das Pendeln des Fahrzeuges wird erreicht, indem abgefragt wird, welche Kurvenform nach dem achten Wechsel an der Reihe sein wird. Die Tabelle dient quasi als Zufallszahlengenerator. Immer wenn an der entsprechenden Stelle eine Linkskurve an der Reihe ist, wird das Hindernis nach rechts bewegt, ansonsten nach links.

```
        lda $d015       Gibt es schon ein Hindernis?
        cmp #$03
        beq $Ja         Ja!
        lda #$03
        sta $d015       Sprite 1 einschalten (Hindernis)
        lda #$ae        Sprite positionieren.
        sta $d002
        lda #$70
        sta $d003
        lda #$80        Form des Sprite
        sta $07f9
Ja      lda $d003       Y-Position Hindernis
        cmp #$e0
```

```
        bne $OK
        lda #$01          Sprite ausschalten
        sta $d015
OK      inc $d003         Setzt Sprite eine Position tiefer
        lda $d003         Position durch 16 teilbar?
        and #$0f
        bne $Nein
        inc $079f         Form verändern
Nein    lda $c056         Pendeln nur bei jedem zweiten
        and #$01          Durchgang.
        beq $Änder
        rts               Rücksprung ins Hauptprogramm
Änder   lda $c057
        adc #$08          Form der achtnächsten Straße
        tax
Schreibt den Accu ins X-Register
        lda $c300,x
        cmp #$29          Linkskurve
        bne $Rechts
        inc $d002         Bewegt Hindernis nach Rechts
        rts               Rücksprung ins Hauptprogramm
Rechts  dec $d002         Bewegt Hindernis nach Links
        rts               Rücksprung ins Hauptprogramm
```

Hier wieder das Basic-Programm:

```
10 IF PEEK (53269) = 3 THEN GOTO 100
20 POKE 53269, 3
30 POKE 53250, 174
40 POKE 53251, 112
50 POKE 1951, 128
100 IF PEEK (53251) "ungleich" 224 THEN GOTO 200
110 POKE 53269, 1
200 POKE 53251, PEEK (53251) + 1
210 A = PEEK (53251)
220 IF A/16 "ungleich" INT (A/16) THEN GOTO 300
230 POKE 1951, PEEK (1951) + 1
```

```
300 IF PEEK (49238) AND 1 THEN GOTO 400
310 RETURN
400 A = PEEK (49920 + PEEK (49239) + 8)
410 IF A "ungleich" 41 THEN GOTO 500
420 POKE 53250, PEEK (53250) + 1
430 RETURN
500 POKE 53250, PEEK (53250) - 1
510 RETURN
```

7. Unterprogramm: Zusammenstöße

Hier wird der Spielverlauf überwacht. Sollte in irgendeiner Situation ein Zusammenstoß aufgetreten sein, so wird dieser registriert und führt zum Verlust der Spielfigur. Die Kontrolle geschieht mit Hilfe der Kollisionsregister. Wird hier in einem der beiden Register das Bit 0 gesetzt, so bedeutet das, daß entweder das Hindernis gerammt oder der Bordstein überfahren wurde. Beides führt aber zum Verlust der Spielfigur.

```
        lda $d01e       Sprite-Sprite Kollision
        ldx #$00
        stx $d01e       Löschen
        cmp #$00        Kollision erfolgt?
        bne $Crash
        lda $d01f       Sprite-Hintergrund Kollision
        stx $d01f       Löschen
        and #$01
        cmp #$01        Kollision erfolgt?
        beq $Crash
        rts             Alles OK, Sprung ins Hauptprogramm
Crash   lda #$01
        sta $d015       Hindernis löschen
        lda #$00
        sta $c05b       Warteschleife initialisieren
        lda #$Warten-Low
        sta $0314       Interrupt-Vektor verändern
        lda #$Warten-High
        sta $0315
        lda #$0d
        sta $07f8       Explosionswolke
        rts             Rücksprung ins Hauptprogramm
Warten  dec $c05b
        beq $Weiter     Warteschleife
        jmp $ea31       Interruptsprung
```

```
Weiter   lda #$ae
         sta $d000     Spielfigur positionieren
         lda #$87
         sta $07f8     Form der Spielfigur
         lda #$00
         sta $d01e     Kollisionsregister löschen
         sta $d01f
         jsr $Vektor   Unterprogramm zur Änderung des
         jmp $ea31     Interrupt Vektors
```

Hier möchte ich von der Regel abweichen und kein Basic-Programm zur Erläuterung beifügen, sondern mir mit einem kurzen Text behelfen.

Der erste Schritt ist das Abfragen der Kollisionsregister. Wenn hier keine Kollision feststellbar ist, dann ist alles OK, und das Programm springt zurück ins Hauptprogramm. Gab es einen Zusammenstoß, dann wird der Interrupt-Vektor geändert und das Programm springt für die nächsten 256 Interrupts in eine Warteschleife. Dieses dauert etwa 4 bis 5 Sekunden. Der letzte Teil sorgt dafür, daß das Programm anschließend weiterlaufen kann. Das Sprite wird positioniert, die Kollisionsregister nochmals gelöscht, da die Explosionswolke seinerseit auch eine Kollision verursacht hat. Als letztes wird das Unterprogramm Vektor angesprungen, um den Interruptvektor wieder auf den alten Wert zu bringen.

8. Unterprogramm: Vektor

Diese Routine sorgt für die Umverlegung des Interrupt-Vektors auf unsere Routine

```
lda $dc0e
and #$fe
sta $dc0e         Interrupt sperren
lda #$Haupt-Low
sta $0314         Verlegen des Interruptvektors
lda #$Haupt-High
sta $0315
lda $dc0e
ora #$01
sta $dc0e         Interrupt entsperren
rts               Rücksprung
```

Auch hier gibt es kein erläuterndes Basic-Programm, sondern eine schriftliche Erklärung.

Das Programm ist in drei Teile gegliedert:

1. Sperrung des Interrupt

2. Verlegung des Interruptvektors

3. Entsperrung des Interrupts

Die Bedeutung der Teile dürfte klar sein. Das Sperren erfolgt aus dem Grunde, daß verhindert werden soll, daß mitten im umPOKEN ein Interrupt ausgelöst wird. Dies hätte schlimme Folgen für unser seelisches Gleichgewicht und sollte deshalb verhindert werden.

Hir ist das komplette Maschinenspracheprogramm. Sie erkennen das Ende jedes Unterprogramms an den drei nop-Befehlen.

```
.,  c000  ad 0e dc   lda $dc0e
.,  c003  29 fe      and #$fe
.,  c005  8d 0e dc   sta $dc0e
.,  c008  a5 01      lda $01
.,  c00a  29 fb      and #$fb
.,  c00c  85 01      sta $01
.,  c00e  ce 15 c0   dec $c015
.,  c011  ce 18 c0   dec $c018
.,  c014  ad 00 cf   lda $cf00
.,  c017  8d 00 07   sta $0700
.,  c01a  ad 15 c0   lda $c015
.,  c01d  d0 ef      bne $c00e
.,  c01f  ce 16 c0   dec $c016
.,  c022  ce 19 c0   dec $c019
.,  c025  ad 16 c0   lda $c016
.,  c028  c9 cf      cmp #$cf
.,  c02a  d0 e2      bne $c00e
.,  c02c  a5 01      lda $01
.,  c02e  09 04      ora #$04
.,  c030  85 01      sta $01
.,  c032  ad 0e dc   lda $dc0e
.,  c035  09 01      ora #$01
.,  c037  8d 0e dc   sta $dc0e
.,  c03a  ad 18 d0   lda $d018
.,  c03d  29 f1      and #$f1
.,  c03f  09 02      ora #$02
.,  c041  8d 18 d0   sta $d018
.,  c044  60         rts
.,  c045  ea         nop
.,  c046  ea         nop
.,  c047  ea         nop
.,  c048  ea         nop
.,  c049  ea         nop
.,  c04a  ea         nop
.,  c04b  ea         nop
```

```
.,  c04c  ea        nop
.,  c04d  ea        nop
.,  c04e  ea        nop
.,  c04f  ea        nop
.,  c050  ea        nop
.,  c051  ea        nop
.,  c052  ea        nop
.,  c053  ea        nop
.,  c054  ea        nop
.,  c055  00        brk
.,  c056  00        brk
.,  c057  00        brk
.,  c058  00        brk
.,  c059  00        brk
.,  c05a  00        brk
.,  c05b  00        brk
.,  c05c  ad 0e dc  lda $dc0e
.,  c05f  29 fe     and #$fe
.,  c061  8d 0e dc  sta $dc0e
.,  c064  a9 00     lda #$00
.,  c066  8d 14 03  sta $0314
.,  c069  a9 c6     lda #$c6
.,  c06b  8d 15 03  sta $0315
.,  c06e  ad 0e dc  lda $dc0e
.,  c071  09 01     ora #$01
.,  c073  8d 0e dc  sta $dc0e
.,  c076  60        rts
.,  c077  ea        nop
.,  c078  ea        nop
.,  c079  ea        nop
.,  c07a  ee 55 c0  inc $c055
.,  c07d  ad 55 c0  lda $c055
.,  c080  c9 08     cmp #$08
.,  c082  f0 09     beq $c08d
.,  c084  c9 10     cmp #$10
.,  c086  f0 11     beq $c099
.,  c088  c9 18     cmp #$18
.,  c08a  f0 1a     beq $c0a6
.,  c08c  60        rts
```

```
.,  c08d  a2 01     ldx #$01
.,  c08f  8e 22 d0  stx $d022
.,  c092  8e 23 d0  stx $d023
.,  c095  20 b9 c0  jsr $c0b9
.,  c098  60        rts
.,  c099  a2 01     ldx #$01
.,  c09b  8e 22 d0  stx $d022
.,  c09e  e8        inx
.,  c09f  8e 23 d0  stx $d023
.,  c0a2  20 b4 c0  jsr $c0b4
.,  c0a5  60        rts
.,  c0a6  a2 00     ldx #$00
.,  c0a8  8e 55 c0  stx $c055
.,  c0ab  e8        inx
.,  c0ac  8e 23 d0  stx $d023
.,  c0af  e8        inx
.,  c0b0  8e 22 d0  stx $d022
.,  c0b3  60        rts
.,  c0b4  a9 09     lda #$09
.,  c0b6  4c bb c0  jmp $c0bb
.,  c0b9  a9 0a     lda #$0a
.,  c0bb  a2 28     ldx #$28
.,  c0bd  9d b8 d9  sta $d9b8,x
.,  c0c0  9d e0 d9  sta $d9e0,x
.,  c0c3  9d 58 da  sta $da58,x
.,  c0c6  9d 80 da  sta $da80,x
.,  c0c9  9d a8 da  sta $daa8,x
.,  c0cc  9d 48 db  sta $db48,x
.,  c0cf  9d f0 d8  sta $d8f0,x
.,  c0d2  9d 18 d9  sta $d918,x
.,  c0d5  9d 40 d9  sta $d940,x
.,  c0d8  9d 68 d9  sta $d968,x
.,  c0db  ca        dex
.,  c0dc  d0 df     bne $c0bd
.,  c0de  60        rts
.,  c0df  ea        nop
.,  c0e0  ea        nop
.,  c0e1  ea        nop
.,  c0e2  ee 56 c0  inc $c056
```

```
.,   c0e5  ad 56 c0   lda $c056
.,   c0e8  c9 40      cmp #$40
.,   c0ea  f0 01      beq $c0ed
.,   c0ec  60         rts
.,   c0ed  a9 00      lda #$00
.,   c0ef  8d 56 c0   sta $c056
.,   c0f2  ee 57 c0   inc $c057
.,   c0f5  ae 57 c0   ldx $c057
.,   c0f8  bd 00 c3   lda $c300,x
.,   c0fb  8d ff c0   sta $c0ff
.,   c0fe  20 00 c1   jsr $c100
.,   c101  60         rts
.,   c102  a2 08      ldx #$08
.,   c104  bd ff c3   lda $c3ff,x
.,   c107  9d ff 04   sta $04ff,x
.,   c10a  bd 07 c4   lda $c407,x
.,   c10d  9d 27 05   sta $0527,x
.,   c110  bd 0f c4   lda $c40f,x
.,   c113  9d 4f 05   sta $054f,x
.,   c116  bd 17 c4   lda $c417,x
.,   c119  9d 77 05   sta $0577,x
.,   c11c  bd 1f c4   lda $c41f,x
.,   c11f  9d 9f 05   sta $059f,x
.,   c122  ca         dex
.,   c123  d0 df      bne $c104
.,   c125  60         rts
.,   c126  a2 08      ldx #$08
.,   c128  bd 27 c4   lda $c427,x
.,   c12b  9d ff 04   sta $04ff,x
.,   c12e  bd 2f c4   lda $c42f,x
.,   c131  9d 27 05   sta $0527,x
.,   c134  bd 37 c4   lda $c437,x
.,   c137  9d 4f 05   sta $054f,x
.,   c13a  bd 3f c4   lda $c43f,x
.,   c13d  9d 77 05   sta $0577,x
.,   c140  bd 47 c4   lda $c447,x
.,   c143  9d 9f 05   sta $059f,x
.,   c146  ca         dex
.,   c147  d0 df      bne $c128
```

```
.,  c149  a2 80     ldx #$80
.,  c14b  bd 7f c4  lda $c47f,x
.,  c14e  9d ff 09  sta $09ff,x
.,  c151  ca        dex
.,  c152  d0 f7     bne $c14b
.,  c154  60        rts
.,  c155  a2 08     ldx #$08
.,  c157  bd 4f c4  lda $c44f,x
.,  c15a  9d ff 04  sta $04ff,x
.,  c15d  bd 57 c4  lda $c457,x
.,  c160  9d 27 05  sta $0527,x
.,  c163  bd 5f c4  lda $c45f,x
.,  c166  9d 4f 05  sta $054f,x
.,  c169  bd 67 c4  lda $c467,x
.,  c16c  9d 77 05  sta $0577,x
.,  c16f  bd 6f c4  lda $c46f,x
.,  c172  9d 9f 05  sta $059f,x
.,  c175  ca        dex
.,  c176  d0 df     bne $c157
.,  c178  a2 88     ldx #$88
.,  c17a  bd ff c4  lda $c4ff,x
.,  c17d  9d ff 09  sta $09ff,x
.,  c180  ca        dex
.,  c181  d0 f7     bne $c17a
.,  c183  60        rts
.,  c184  ea        nop
.,  c185  ea        nop
.,  c186  ea        nop
.,  c187  ae 57 c0  ldx $c057
.,  c18a  bd 00 c3  lda $c300,x
.,  c18d  c9 02     cmp #$02
.,  c18f  d0 01     bne $c192
.,  c191  60        rts
.,  c192  c9 26     cmp #$26
.,  c194  d0 42     bne $c1d8
.,  c196  ad 56 c0  lda $c056
.,  c199  f0 01     beq $c19c
.,  c19b  60        rts
.,  c19c  ad 4f 04  lda $044f
```

```
.,  c19f  8d 58 c0   sta $c058
.,  c1a2  ad 77 04   lda $0477
.,  c1a5  8d 59 c0   sta $c059
.,  c1a8  ad 9f 04   lda $049f
.,  c1ab  8d 5a c0   sta $c05a
.,  c1ae  a2 27      ldx #$27
.,  c1b0  bd 27 04   lda $0427,x
.,  c1b3  9d 28 04   sta $0428,x
.,  c1b6  bd 4f 04   lda $044f,x
.,  c1b9  9d 50 04   sta $0450,x
.,  c1bc  bd 77 04   lda $0477,x
.,  c1bf  9d 78 04   sta $0478,x
.,  c1c2  ca         dex
.,  c1c3  d0 eb      bne $c1b0
.,  c1c5  ad 58 c0   lda $c058
.,  c1c8  8d 28 04   sta $0428
.,  c1cb  ad 59 c0   lda $c059
.,  c1ce  8d 50 04   sta $0450
.,  c1d1  ad 5a c0   lda $c05a
.,  c1d4  8d 78 04   sta $0478
.,  c1d7  60         rts
.,  c1d8  ad 56 c0   lda $c056
.,  c1db  f0 01      beq $c1de
.,  c1dd  60         rts
.,  c1de  ad 28 04   lda $0428
.,  c1e1  8d 58 c0   sta $c058
.,  c1e4  ad 50 04   lda $0450
.,  c1e7  8d 59 c0   sta $c059
.,  c1ea  ad 78 04   lda $0478
.,  c1ed  8d 5a c0   sta $c05a
.,  c1f0  a2 00      ldx #$00
.,  c1f2  bd 29 04   lda $0429,x
.,  c1f5  9d 28 04   sta $0428,x
.,  c1f8  bd 51 04   lda $0451,x
.,  c1fb  9d 50 04   sta $0450,x
.,  c1fe  bd 79 04   lda $0479,x
.,  c201  9d 78 04   sta $0478,x
.,  c204  e8         inx
.,  c205  e0 27      cpx #$27
```

```
.,  c207  d0 e9     bne $c1f2
.,  c209  ad 58 c0  lda $c058
.,  c20c  8d 4f 04  sta $044f
.,  c20f  ad 59 c0  lda $c059
.,  c212  8d 77 04  sta $0477
.,  c215  ad 5a c0  lda $c05a
.,  c218  8d 9f 04  sta $049f
.,  c21b  60        rts
.,  c21c  ea        nop
.,  c21d  ea        nop
.,  c21e  ea        nop
.,  c21f  ae 57 c0  ldx $c057
.,  c222  bd 00 c3  lda $c300,x
.,  c225  c9 02     cmp #$02
.,  c227  d0 01     bne $c22a
.,  c229  60        rts
.,  c22a  ad 56 c0  lda $c056
.,  c22d  29 01     and #$01
.,  c22f  c9 01     cmp #$01
.,  c231  f0 01     beq $c234
.,  c233  60        rts
.,  c234  bd 00 c3  lda $c300,x
.,  c237  c9 26     cmp #$26
.,  c239  d0 04     bne $c23f
.,  c23b  ee 00 d0  inc $d000
.,  c23e  60        rts
.,  c23f  ce 00 d0  dec $d000
.,  c242  60        rts
.,  c243  ea        nop
.,  c244  ea        nop
.,  c245  ea        nop
.,  c246  ad 15 d0  lda $d015
.,  c249  c9 03     cmp #$03
.,  c24b  f0 14     beq $c261
.,  c24d  a9 03     lda #$03
.,  c24f  8d 15 d0  sta $d015
.,  c252  a9 ae     lda #$ae
.,  c254  8d 02 d0  sta $d002
.,  c257  a9 70     lda #$70
```

```
.,    c259   8d 03 d0   sta $d003
.,    c25c   a9 80      lda #$80
.,    c25e   8d f9 07   sta $07f9
.,    c261   ad 03 d0   lda $d003
.,    c264   c9 e0      cmp #$e0
.,    c266   d0 05      bne $c26d
.,    c268   a9 01      lda #$01
.,    c26a   8d 15 d0   sta $d015
.,    c26d   ee 03 d0   inc $d003
.,    c270   ad 03 d0   lda $d003
.,    c273   29 0f      and #$0f
.,    c275   d0 03      bne $c27a
.,    c277   ee f9 07   inc $07f9
.,    c27a   ad 56 c0   lda $c056
.,    c27d   29 01      and #$01
.,    c27f   f0 01      beq $c282
.,    c281   60         rts
.,    c282   ad 57 c0   lda $c057
.,    c285   69 08      adc #$08
.,    c287   aa         tax
.,    c288   bd 00 c3   lda $c300,x
.,    c28b   c9 26      cmp #$26
.,    c28d   d0 04      bne $c293
.,    c28f   ee 02 d0   inc $d002
.,    c292   60         rts
.,    c293   ce 02 d0   dec $d002
.,    c296   60         rts
.,    c297   ea         nop
.,    c298   ea         nop
.,    c299   ea         nop
.,    c29a   ad 1e d0   lda $d01e
.,    c29d   a2 00      ldx #$00
.,    c29f   8e 1e d0   stx $d01e
.,    c2a2   c9 00      cmp #$00
.,    c2a4   d0 0d      bne $c2b3
.,    c2a6   ad 1f d0   lda $d01f
.,    c2a9   8e 1f d0   stx $d01f
.,    c2ac   29 01      and #$01
.,    c2ae   c9 01      cmp #$01
```

```
.,   c2b0  f0 01      beq $c2b3
.,   c2b2  60         rts
.,   c2b3  a9 01      lda #$01
.,   c2b5  8d 15 d0   sta $d015
.,   c2b8  a9 00      lda #$00
.,   c2ba  8d 5b c0   sta $c05b
.,   c2bd  a9 cd      lda #$cd
.,   c2bf  8d 14 03   sta $0314
.,   c2c2  a9 c2      lda #$c2
.,   c2c4  8d 15 03   sta $0315
.,   c2c7  a9 0d      lda #$0d
.,   c2c9  8d f8 07   sta $07f8
.,   c2cc  60         rts
.,   c2cd  ce 5b c0   dec $c05b
.,   c2d0  f0 03      beq $c2d5
.,   c2d2  4c 31 ea   jmp $ea31
.,   c2d5  a9 ae      lda #$ae
.,   c2d7  8d 00 d0   sta $d000
.,   c2da  a9 87      lda #$87
.,   c2dc  8d f8 07   sta $07f8
.,   c2df  a9 00      lda #$00
.,   c2e1  8d 1e d0   sta $d01e
.,   c2e4  8d 1f d0   sta $d01f
.,   c2e7  20 5c c0   jsr $c05c
.,   c2ea  4c 31 ea   jmp $ea31
.,   c2ed  00         brk
.,   c2ee  00         brk
.,   c2ef  00         brk
.,   c2f0  00         brk
```

Das Hauptprogramm

Dies ist das eigentliche Hauptprogramm. Diese Routine wird durch den Interrupt angesprungen und springt dann die einzelnen Unterprogramme an.

```
.,  c600  48        pha
.,  c601  98        tya
.,  c602  48        pha
.,  c603  8a        txa
.,  c604  48        pha
.,  c605  a9 e0     lda #$e0
.,  c607  8d 02 dc  sta $dc02
.,  c60a  ad 00 dc  lda $dc00
.,  c60d  29 04     and #$04
.,  c60f  d0 06     bne $c617
.,  c611  ce 00 d0  dec $d000
.,  c614  4c 21 c6  jmp $c621
.,  c617  ad 00 dc  lda $dc00
.,  c61a  29 08     and #$08
.,  c61c  d0 03     bne $c621
.,  c61e  ee 00 d0  inc $d000
.,  c621  a9 ff     lda #$ff
.,  c623  8d 02 dc  sta $dc02
.,  c626  20 7a c0  jsr $c07a
.,  c629  20 e2 c0  jsr $c0e2
.,  c62c  20 87 c1  jsr $c187
.,  c62f  20 1f c2  jsr $c21f
.,  c632  20 46 c2  jsr $c246
.,  c635  20 9a c2  jsr $c29a
.,  c638  68        pla
.,  c639  aa        tax
.,  c63a  68        pla
.,  c63b  a8        tay
.,  c63c  68        pla
.,  c63d  4c 31 ea  jmp $ea31
.
```

14.5.4. Bird

In diesem 4. Programm möchte ich allen denen einen Gefallen tun, die bei Weltraumspielen immer ein gewisses Zucken in der Hand haben.

Sie kennen doch sicher Spiele, bei denen am oberen Bildschirmrand ein Schwarm bizarr geformter Wesen sein Unwesen treibt, während am unteren Bildschirmrand ein Raumschiff hin und her saust. Ein solches Spiel soll hier beschrieben werden.

Die Spielidee:

Auf dem oberen Teil des Bildschirms soll sich ein Schwarm von 21 Vögeln bewegen. Diese Vögel bewegen sich jeweils von einem Bildschirmrand zum anderen. Am unteren Bildschirmrand läßt sich per Joystick ein Raumschiff hin und her bewegen, auf Knopfdruck feuert das Raumschiff einen Schuß ab. Bei einem Treffer - er zählt im übrigen nur, wenn die Mitte des Vogels getroffen wurde - wird der entsprechende Vogel gelöscht und 10 Punkte werden zu dem Score dazuaddiert. Sind alle Vögel getroffen, so wird ein neuer Schwarm abgebildet und die Sache geht von vorne los.

Das klingt alles schon ganz verlockend, hat nur einen Haken: Bis jetzt müssen es sich die Vögel gefallen lassen, als Zielscheiben zu dienen. Das ist natürlich ein unbefriedigender Zustand, und so haben sie sich etwas ganz Mieses ausgedacht:

Ab und zu schicken sie einen "Kumpel" auf die Reise, um dem Raumschiff das Leben schwer zu machen. Dieser letzte Vogel hat zudem auch noch die Angewohnheit, mitten im Fluge Eier zu legen.

So, lassen wir den etwas lockeren Ton und kommen wir zu einer genaueren Beschreibung des Spieles.

Der Hintergrund:

Als Hintergrund muß man bei diesem Spiel den Vogelschwarm am "Himmel" werten. Er besteht aus selbstdefinierten Zeichen (keine Angst, es sind lediglich 6 Zeichen umzudefinieren). Der Schwarm besteht aus 21 Vögeln und bewegt sich am oberen Bildschirmrand hin und her.

Die Spielfigur:

Die Figur, mit der wir uns im Spiel bewegen, ist ein Raumschiff am unteren Rand des Spielfeldes. Man kann das Raumschiff über die gesamte Breite des Bildschirms bewegen; das heißt für uns als Programmierer, daß wir darauf achten müssen, wann das 9. Bit der Sprite-Koordinate gesetzt bzw. wieder gelöscht werden muß (siehe auch das Kapitel über die Sprite-Programmierung).

Um die Vögel vom Bildschirm "entfernen" zu können, brauchen wir natürlich ein Mittel: Einen Schuß.

Der Schuß:

Der Schuß wird durch einen Druck auf die Feuertaste des Joysticks ausgelöst. Er eilt dann in Achter-Schritten dem oberen Bildschirmrand entgegen und wird erst gelöscht, wenn er entweder einen Treffer erzielt oder aber den oberen Bildschirmrand erreicht. Als Treffer gewertet werden folgende drei Ereignisse:

1. Kollision mit einem Vogel aus dem Schwarm. Hier gibt es allerdings noch die Einschränkung, daß der Vogel sozusagen in der Mitte getroffen werden muß: ein Treffer am Flügel wird ignoriert. Ein Treffer dieser Art ergibt 10 Punkte zugunsten des Spielers.

2. Kollision mit einem Vogel, der sich frei über dem Bildschirm bewegt. Hier zählt ein Treffer an beliebiger Stelle. Da es ungleich schwerer ist, ein bewegliches Ziel zu treffen, wird ein solches Ereignis auch ein bißchen besser bewertet. 50 Punkte werden für einen solchen Treffer gegeben.

3. Kollision mit einem Ei. In diesem Spiel ist es möglich, die Eier eines Vogels abzuschießen, ein solcher Treffer wird ebenfalls mit 50 Punkten bewertet.

Das Hindernis:

Um die Sache zu erschweren, kommt beinahe ständig ein Vogel vom Himmel, lediglich mit der Aufgabe, das Raumschiff zu treffen.

Etwas konkreter sieht die Sache so aus: Ein Sprite zieht im Zick-Zack seine Bahn auf dem Bildschirm. Dabei richtet sich die Position des Sprites teilweise nach der Spielfigur, andererseits wird sie auch durch den Zufall bestimmt.

Auf dem Weg über den Bildschirm legt der Vogel Eier. Diese fallen auf dem kürzesten Wege zum Boden und versuchen ebenfalls, das Raumschiff zu treffen. Je tiefer der Vogel auf dem Bildschirm ist, desto häufiger legt er Eier.

Spielzustände:

Folgende Zustände führen zu einer gesonderten Behandlung.

Positive Ereignisse:

1. Treffer eines Vogels im Schwarm
2. Treffer eines beweglichen Vogels
3. Treffer eines Eies
4. Treffer des letzten Vogels eines Schwarmes

Die Behandlung der Zustände 1 bis 3 erfolgte schon im Laufe des Textes. Zustand 4 wird wie folgt behandelt:

Mit dem Treffer des letzten Vogels reagiert das Programm mit dem Aufbau eines neuen Schwarmes. Weitere Auswirkungen gibt es nicht.

Negative Ereignisse

1. Kollision eines Vogels mit dem Raumschiff
2. Kollision eines Eies mit dem Raumschiff
3. Dritte Kollision eines Hindernisses mit dem Raumschiff.

Bei der Behandlung der Ereignisse eins und zwei gibt es keine Unterscheidung: Sämtliche Sprites werden gelöscht, das Raumschiff neu positioniert, und die Anzahl der Raumschiffe sowohl im Programm als auch auf dem Bildschirm verändert.

Mit dem Verlust des dritten Raumschiffes (3.Ereignis) wird zuerst das zuvor beschriebene ausgeführt. Damit hat sich die Sache allerdings noch nicht erledigt: Der Interrupt-Vektor wird wieder auf das normale Ziel gerichtet, das Programm somit angehalten, gleichzeitig erscheint auf dem Bildschirm

die Aufschrift "Game Over". Ein neues Spiel wird durch einen Tastendruck gestartet.

Das Programm:

Wie schon üblich, möchte ich auch hier zuerst das Basic-Programm beschreiben.

Dieses Programm besticht, wie das vorherige auch, durch eine enorme Anzahl von DATA-Statements.

Um das Programm erfolgreich laufen lassen zu können, muß man sich auch hier Speicherplatz reservieren. Dies geschieht durch:

POKE 44,24: POKE 6144, 0: NEW

Nach dieser Prozedur kann das Programm gestartet werden und wird seinen Dienst verrichten.

Listing Bird:

```
1 PRINT"[illegible]";:S=10
10 DATA 169, 1, 133, 43, 169, 24, 133, 4
4, 716
11 DATA 169, 0, 141, 0, 24, 234, 234, 23
4, 1036
12 DATA 173, 14, 220, 41, 254, 141, 14,
220, 1077
13 DATA 165, 1, 41, 251, 133, 1, 206, 37
, 835
14 DATA 192, 206, 40, 192, 173, 255, 223
, 141, 1422
15 DATA 255, 23, 173, 37, 192, 208, 239,
 206, 1333
16 DATA 38, 192, 206, 41, 192, 173, 38,
192, 1072
17 DATA 201, 207, 208, 226, 165, 1, 9, 4
, 1021
18 DATA 133, 1, 173, 14, 220, 9, 1, 141,
 692
19 DATA 14, 220, 173, 24, 208, 41, 241,
9, 930
20 DATA 2, 141, 24, 208, 96, 0, 0, 0, 47
1
25 FORI=0TO10
26 FORR=0TO7
27 READA:POKE49152+8*I+R,A:X=X+A:NEXTR
28 READA:IFA<>XTHEN5000
29 X=0:S=S+1:NEXTI:S=100
35 SYS49152
40 POKE53272,(PEEK(53272)AND241)OR2
100 DATA 0, 128, 224, 254, 127, 127, 63,
 31, 954
101 DATA 48, 232, 188, 60, 255, 255, 255
, 255, 1548
```

```
102 DATA 0, 1, 7, 127, 254, 254, 252, 24
8, 1143
103 DATA 7, 3, 0, 0, 1, 3, 14, 26, 54
104 DATA 255, 255, 255, 187, 153, 60, 60
, 126, 1351
105 DATA 224, 192, 0, 0, 128, 192, 112,
88, 936
110 FORI=0TO5
111 FORR=0TO7
112 READA:POKE2560+8*I+R,A:X=X+A:NEXTR
113 READA:IFA<>XTHEN5000
114 X=0:S=S+1:NEXTI:S=200
200 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
201 DATA 0, 0, 16, 0, 0, 16, 0, 0, 32
202 DATA 56, 0, 0, 56, 0, 0, 124, 0, 236
203 DATA 0, 254, 0, 0, 186, 0, 0, 56, 49
6
204 DATA 0, 32, 124, 8, 112, 124, 28, 11
2, 540
205 DATA 124, 28, 248, 254, 62, 248, 254
, 62, 1280
206 DATA 248, 254, 62, 255, 255, 254, 25
5, 255, 1838
207 DATA 254, 248, 238, 62, 80, 68, 20,
0, 970
208 DATA 0, 24, 0, 0, 60, 0, 0, 60, 144
209 DATA 0, 0, 60, 0, 0, 60, 0, 0, 120
210 DATA 60, 0, 0, 60, 0, 0, 24, 0, 144
211 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
212 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
213 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
214 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
215 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
216 DATA 0, 34, 0, 0, 54, 0, 0, 62, 150
217 DATA 0, 0, 62, 0, 0, 28, 0, 0, 90
218 DATA 28, 0, 0, 28, 0, 0, 127, 0, 183
219 DATA 0, 255, 128, 1, 255, 192, 67, 2
55, 1153
```

```
220 DATA 225, 103, 255, 243, 63, 255, 25
4, 31, 1429
221 DATA 255, 252, 15, 255, 248, 7, 62,
112, 1206
222 DATA 0, 62, 0, 0, 28, 0, 0, 28, 118
223 DATA 0, 0, 8, 0, 0, 8, 0, 0, 16
230 FORI=0TO23
231 FORR=0TO7
232 READA:POKE832+8*I+R,A:X=X+A:NEXTR
233 READA:IFA<>XTHEN5000
234 X=0:S=S+1:NEXTI:S=300
300 DATA 173, 14, 220, 41, 254, 141, 14,
 220, 1077
301 DATA 169, 0, 141, 20, 3, 169, 198, 1
41, 841
302 DATA 21, 3, 173, 14, 220, 9, 1, 141,
 582
303 DATA 14, 220, 234, 234, 234, 238, 0,
 192, 1366
304 DATA 173, 0, 192, 201, 8, 240, 1, 96
, 911
305 DATA 169, 224, 141, 2, 220, 173, 0,
220, 1149
306 DATA 141, 1, 192, 162, 255, 142, 2,
220, 1115
307 DATA 232, 142, 0, 192, 96, 234, 234,
 234, 1364
308 DATA 173, 1, 192, 41, 4, 208, 28, 20
6, 853
309 DATA 0, 208, 173, 0, 208, 201, 255,
208, 1253
310 DATA 8, 173, 16, 208, 73, 1, 141, 16
, 636
311 DATA 208, 173, 0, 192, 201, 4, 208,
3, 989
312 DATA 206, 2, 192, 173, 1, 192, 41, 8
, 815
```

```
313 DATA 208, 23, 238, 0, 208, 208, 8, 1
73, 1066
314 DATA 16, 208, 73, 1, 141, 16, 208, 1
73, 836
315 DATA 0, 192, 201, 4, 208, 3, 238, 2,
 848
316 DATA 192, 173, 1, 192, 41, 16, 240,
1, 856
317 DATA 96, 173, 21, 208, 41, 2, 240, 1
, 782
318 DATA 96, 173, 21, 208, 9, 2, 141, 21
, 671
319 DATA 208, 173, 0, 208, 141, 2, 208,
169, 1109
320 DATA 192, 141, 3, 208, 173, 16, 208,
 41, 982
321 DATA 1, 201, 1, 208, 8, 173, 16, 208
, 816
322 DATA 9, 2, 141, 16, 208, 173, 2, 192
, 743
323 DATA 141, 3, 192, 169, 17, 141, 4, 1
92, 859
324 DATA 96, 234, 234, 234, 173, 21, 208
, 41, 1241
325 DATA 2, 201, 2, 240, 1, 96, 162, 8,
712
326 DATA 206, 3, 208, 202, 208, 250, 206
, 4, 1287
327 DATA 192, 173, 3, 208, 240, 1, 96, 1
73, 1086
328 DATA 21, 208, 41, 253, 141, 21, 208,
 173, 1066
329 DATA 16, 208, 41, 253, 141, 16, 208,
 96, 979
330 DATA 234, 234, 234, 173, 0, 208, 201
, 24, 1308
331 DATA 208, 18, 173, 16, 208, 41, 1, 2
01, 866
```

```
332 DATA 1, 240, 9, 173, 1, 192, 9, 4, 6
29
333 DATA 141, 1, 192, 96, 173, 0, 208, 2
01, 1012
334 DATA 64, 240, 1, 96, 173, 16, 208, 4
1, 839
335 DATA 1, 240, 8, 173, 1, 192, 9, 8, 6
32
336 DATA 141, 1, 192, 96, 234, 234, 234,
 173, 1305
337 DATA 4, 192, 201, 0, 240, 83, 234, 1
73, 1127
338 DATA 21, 208, 41, 2, 201, 2, 240, 1,
 716
339 DATA 96, 174, 4, 192, 172, 3, 192, 2
4, 857
340 DATA 32, 240, 255, 164, 211, 177, 20
9, 201, 1489
341 DATA 68, 240, 1, 96, 169, 32, 145, 2
09, 960
342 DATA 200, 145, 209, 136, 136, 145, 2
09, 172, 1352
343 DATA 3, 192, 174, 4, 192, 202, 24, 3
2, 823
344 DATA 240, 255, 164, 211, 169, 32, 13
6, 145, 1352
345 DATA 209, 200, 145, 209, 200, 145, 2
09, 238, 1555
346 DATA 7, 192, 173, 7, 192, 201, 21, 2
08, 1001
347 DATA 3, 32, 237, 194, 169, 1, 32, 32
, 700
348 DATA 195, 173, 21, 208, 41, 253, 141
, 21, 1053
349 DATA 208, 173, 16, 208, 41, 253, 141
, 16, 1056
350 DATA 208, 96, 234, 234, 234, 238, 5,
 192, 1441
```

```
351 DATA 173, 5, 192, 201, 48, 240, 1, 9
6, 956
352 DATA 169, 0, 141, 5, 192, 76, 168, 1
94, 945
353 DATA 238, 6, 192, 173, 6, 192, 201,
11, 1019
354 DATA 208, 10, 169, 0, 141, 6, 192, 1
69, 895
355 DATA 200, 141, 166, 194, 162, 255, 1
89, 255, 1562
356 DATA 3, 157, 0, 4, 202, 208, 247, 96
, 917
357 DATA 238, 6, 192, 173, 6, 192, 201,
11, 1019
358 DATA 208, 10, 169, 0, 141, 6, 192, 1
69, 895
359 DATA 168, 141, 166, 194, 162, 0, 189
, 0, 1020
360 DATA 4, 157, 255, 3, 232, 224, 255,
208, 1338
361 DATA 245, 96, 234, 234, 234, 169, 0,
 141, 1353
362 DATA 7, 192, 162, 29, 189, 15, 192,
157, 943
363 DATA 0, 4, 157, 80, 4, 157, 160, 4,
566
364 DATA 189, 43, 192, 157, 40, 4, 157,
120, 902
365 DATA 4, 157, 200, 4, 202, 208, 229,
169, 1173
366 DATA 0, 141, 5, 192, 141, 6, 192, 16
9, 846
367 DATA 168, 141, 166, 194, 96, 234, 23
4, 234, 1467
368 DATA 248, 162, 5, 24, 125, 80, 192,
157, 993
369 DATA 80, 192, 41, 240, 240, 6, 169,
1, 969
```

```
370 DATA 202, 16, 240, 216, 162, 5, 189,
 80, 1110
371 DATA 192, 41, 15, 157, 80, 192, 24,
105, 806
372 DATA 48, 157, 201, 7, 202, 208, 239,
 96, 1158
373 DATA 234, 234, 234, 173, 21, 208, 41
, 4, 1149
374 DATA 201, 4, 208, 1, 96, 173, 43, 17
6, 902
375 DATA 238, 86, 195, 141, 4, 208, 169,
 0, 1041
376 DATA 141, 5, 208, 173, 21, 208, 9, 4
, 769
377 DATA 141, 21, 208, 96, 234, 234, 234
, 238, 1406
378 DATA 8, 192, 173, 8, 192, 201, 64, 2
40, 1078
379 DATA 1, 96, 169, 0, 141, 8, 192, 173
, 780
380 DATA 30, 160, 238, 128, 195, 205, 0,
 208, 1164
381 DATA 48, 6, 169, 1, 141, 9, 192, 96,
 662
382 DATA 169, 2, 141, 9, 192, 96, 234, 2
34, 1077
383 DATA 234, 238, 5, 208, 173, 9, 192,
201, 1260
384 DATA 1, 208, 19, 206, 4, 208, 173, 4
, 823
385 DATA 208, 201, 255, 208, 8, 173, 16,
 208, 1277
386 DATA 73, 4, 141, 16, 208, 96, 238, 4
, 780
387 DATA 208, 208, 8, 173, 16, 208, 73,
4, 898
388 DATA 141, 16, 208, 96, 234, 234, 234
, 173, 1336
```

```
389 DATA 16, 208, 41, 4, 201, 4, 240, 13
, 727
390 DATA 173, 4, 208, 201, 24, 208, 5, 1
69, 992
391 DATA 0, 141, 9, 192, 96, 173, 4, 208
, 823
392 DATA 201, 64, 208, 5, 169, 1, 141, 9
, 798
393 DATA 192, 96, 234, 234, 234, 173, 21
, 208, 1392
394 DATA 41, 8, 201, 8, 240, 37, 173, 4,
 712
395 DATA 208, 141, 6, 208, 173, 16, 208,
 41, 1001
396 DATA 4, 201, 4, 208, 8, 173, 16, 208
, 822
397 DATA 9, 8, 141, 16, 208, 173, 5, 208
, 768
398 DATA 141, 7, 208, 173, 21, 208, 9, 8
, 775
399 DATA 141, 21, 208, 238, 7, 208, 238,
 7, 1068
400 DATA 208, 173, 7, 208, 41, 254, 201,
 0, 1092
401 DATA 240, 1, 96, 173, 21, 208, 41, 2
47, 1027
402 DATA 141, 21, 208, 173, 16, 208, 41,
 247, 1055
403 DATA 141, 16, 208, 96, 234, 234, 234
, 173, 1336
404 DATA 30, 208, 141, 12, 192, 169, 0,
141, 893
405 DATA 30, 208, 173, 12, 192, 41, 1, 2
01, 858
406 DATA 1, 208, 50, 169, 1, 141, 21, 20
8, 799
407 DATA 169, 20, 141, 2, 192, 169, 176,
 141, 1010
```

```
408 DATA 0, 208, 169, 0, 141, 16, 208, 2
06, 948
409 DATA 10, 192, 208, 16, 169, 1, 141,
11, 748
410 DATA 192, 169, 49, 141, 20, 3, 169,
234, 977
411 DATA 141, 21, 3, 96, 174, 10, 192, 1
69, 806
412 DATA 32, 157, 220, 7, 96, 173, 12, 1
92, 889
413 DATA 201, 12, 208, 1, 96, 201, 6, 20
8, 933
414 DATA 22, 173, 21, 208, 41, 249, 141,
 21, 876
415 DATA 208, 173, 16, 208, 41, 249, 141
, 16, 1052
416 DATA 208, 169, 5, 32, 32, 195, 96, 1
73, 910
417 DATA 12, 192, 201, 10, 240, 1, 96, 1
73, 925
418 DATA 21, 208, 41, 245, 141, 21, 208,
 173, 1058
419 DATA 16, 208, 41, 245, 141, 16, 208,
 169, 1044
420 DATA 5, 32, 32, 195, 96, 234, 234, 2
34, 1062
425 FORI=0TO120
426 FORR=0TO7
427 READA:POKE49408+8*I+R,A:X=X+A:NEXTR
428 READA:IFA<>XTHEN5000
429 X=0:S=S+1:NEXTI:S=450
450 DATA 72, 152, 72, 138, 72, 32, 29, 1
93, 760
451 DATA 32, 243, 193, 32, 64, 193, 32,
196, 985
452 DATA 193, 32, 39, 194, 32, 149, 194,
 32, 865
```

```
453 DATA 75, 195, 32, 111, 195, 32, 153,
 195, 988
454 DATA 32, 199, 195, 32, 237, 195, 32,
 63, 985
455 DATA 196, 104, 170, 104, 168, 104, 7
6, 49, 971
456 DATA 234, 0, 0, 0, 0, 0, 0, 0, 234
460 FORI=0TO6
461 FORR=0TO7
462 READA:POKE50688+8*I+R,A:X=X+A:NEXTR
463 READA:IFA<>XTHEN5000
464 X=0:S=S+1:NEXTI
500 V=53248:S=49152
502 POKEV+21,1
504 POKEV,176:POKES+2,20
506 POKES+5,0
508 POKES+6,0
510 POKES+7,0
512 POKE49830,168
514 POKEV+1,200
516 POKE2040,13:POKE2041,14:POKE2042,15:
POKE2043,14
518 POKE53281,14
520 PRINT"                              ";
522 PRINT"                              ";
524 PRINT"                              ";
526 PRINT"                              ";
528 PRINT"                              ";
530 PRINT"                              "
532 FORI=0TO6
534 POKE49152+16+4*I,32
536 POKE49152+17+4*I,64
538 POKE49152+18+4*I,65
540 POKE49152+19+4*I,66
542 POKE49152+44+4*I,32
544 POKE49152+45+4*I,67
546 POKE49152+46+4*I,68
548 POKE49152+47+4*I,69
```

```
550 NEXTI
552 POKE49152+72,32:POKE49162,3
554 FORI=0TO5:POKE49152+80+I,0:NEXT
556 FORI=0TO5:POKE53248+39+I,1:NEXT
558 PRINT"SCORE:       00
000    SCHIFFE: ++";
560 SYS49408
562 POKE49163,0:WAIT49163,1
564 PRINT""
566 PRINT"GAME
OVER"
568 PRINT"TASTE  DRUECKEN"
570 POKE198,0:WAIT198,1:RUN
5000 PRINT"DATA FEHLER IN ZEILE";S;"!!!!
!":END
```

Das Programm setzt sich aus insgesamt 5 DATA-Blöcken zusammen. Jeder DATA-Block wird unmittelbar von dem zuständigen Lader gefolgt.

Auch hier besitzt jede DATA-Zeile neun Bytes, wobei das neunte Byte als Prüfsumme dient. Falls Sie das Programm zeilengetreu abtippen wird sich das Programm mit der Meldung "DATA Fehler in Zeile XX" melden. Der Fehler ist dann entweder in der betreffenden Zeile selbst oder aber in der vorhergehenden zu finden.

DATA-Block 1

Dieser Block beginnt an der Programmzeile 10 und endet bei der Zeile 40. Dieser Block sorgt für die Umverlegung des Character-Generators und die Umschaltung des Zeichensatzes.

DATA-Block 2

Er wird durch die Zeilen 100 bis 114 gebildet, durch ihn werden die Zeichen Shift *, Shift A, ... bis Shift E umdefiniert.

DATA-Block 3

Zeile 200 bis 234. Er ist verantwortlich für das Aussehen der Sprites. Die Informationen werden in den Kassettenpuffer ab 832 geschrieben.

DATA Block 4

Der mit Abstand größte Block umfaßt die Zeilen 300 bis einschließlich 429. Hier sind die Unterprogramme für die erweiterte Interruptroutine zusammengefaßt.

DATA-Block 5

Er geht von Zeile 450 bis 464 und umfaßt das eigentliche Maschinenspracheprogramm. Dieses Programm wird durch den Interrupt angesprungen und durchlaufen. Das Programm selber besteht nur aus Sprüngen zu den verschiedenen Unterroutinen (DATA-Block 4), und dem abschließenden Sprung zum eigentlichen Interrupt.

Richtig interessant wird das Programm erst jetzt.

Bis zur Zeile 530 werden der Bildschirm vorbereitet und für das Programm wichtige Pointer gesetzt. Der Befehl in der Zeile 512 ist wichtig für einen zweiten Programmdurchlauf: Er sorgt dafür, daß der Vogelschwarm zuerst nach rechts pendelt.

Die Zeilen 532 bis 558 sorgen dafür, daß dem Programm diverse Byte-Tabellen zur Verfügung stehen, etwa für den Aufbau eines neuen Schwarmes, oder aber, daß die Scoreanzeige mit 0 beginnt.

Zeile 560 startet das Maschinenspracheprogramm.

Zeile 562 bis 570 sorgen für die Wiederholung des Spieles, nachdem das Raumschiff zum dritten Male zerstört wurde. Ausschlaggebend für den Start dieser Routine ist der WAIT Befehl in Zeile 570. Hier "hängt" das Programm, bis dieses Byte durch das Maschinenspracheprogramm gesetzt wird.

Zeile 5000 sorgt für eine Fehlermeldung bei einem Tippfehler in den DATAs.

Auch in diesem Programm muß ein Teil des RAMs geschützt werden, in diesem Fall wird der BASIC-Start an die Adresse 6144 verlegt. Nach dem Eintippen speichern wir also erst das Programm ab und tippen

POKE 44, 24: NEW

ein. Danach laden wir erneut und können dann starten.

Der Maschinensprache-Teil:

Der Maschinensprache-Teil besteht aus insgesammt 16 Teilen plus Hauptprogramm. Einiges ist uns schon aus den vorherigen Programmen bekannt oder ähnelt diesen stark. Die Beschreibung dieser Teile wird also insgesamt nicht so ausführlich geschehen, da Sie sich nun schon mehr unter den Routinen vorstellen können

Teil 1:

Der Teil 1 sorgt für eine Kopie des Character-Generators im RAM ab der Adresse 2048. Er ist identisch mit der Routine von dem Programm "Street", eine Erklärung erübrigt sich also.

Teil 2:

Auch diesen Teil kennen wir schon, er verbiegt den Interrupt-Vektor

```
lda $dc0e
and #$fe
sta $dc0e       Sperrt den Interrupt
lda #$00
sta $0314
lda #$c6
sta $0315       Verlegt den Interrupt
lda $dc0e
ora #$01
sta $dc0e
```

Gibt den Interrupt wieder frei

Teil 3:

Auch hier etwas bereits Bekanntes. Dieser Teil fragt alle acht Interrupts den Joystick-Port ab und speichert den Inhalt an der Adresse $c001 zwischen. Der Achterhrythmus sorgt dafür, daß das Raumschiff jeweils genau in einer 8 * 8 Matrix auf dem Bildschirm steht, also genau unter einem Zeichen.

```
        inc $c000
        lda $c000
        cmp #$08        Achterrhythmus
        beq $Abfrage
        rts             Rücksprung ins Hauptprogramm
Abfrage lda #$e0
        sta $dc02       Schaltet Port 2 ein
        lda $dc00
        sta $c001       Zwischenspeichern des Inhalts
        ldx #$ff
        stx $dc02       Schaltet Port 2 aus
        inx
        ldx $c000       Initialisiert den Zähler für die
                        Abfrage
        rts             Rücksprung ins Hauptprogramm
```

Teil 4:

Dieser Teil sorgt für die Bewegung des Sprites und schaltet auch bei Bedarf den Schuß ein.

```
        lda $c001
        and #$04
        bne $Rechts
        dec $d000       Bewegt Raumschiff nach links
        lda $d000
```

```
        cmp #$ff        Muß "Bit 9" verändert werden?
        bne $Weiter
        lda $d010
        eor #$01        Verändert "Bit 9" der X-Position
        sta $d010
Weiter  lda $c000
        cmp #$04
        bne $Rechts
        dec $c002       Spalte in der sich das Sprite auf
                        dem Bildschirm befindet
Rechts  lda $c001
        and #$08
        bne $Schuß
        inc $d000       Bewegt Raumschiff nach rechts
        bne $Weiter2    Muß "Bit 9" verändert werden?
        lda $d010
        eor #$01
        sta $d010       Verändert "Bit 9" der X-Position
Weiter2 lda $c000
        cmp #$04
        bne $Schuß
        inc $c002       Spalte, in der sich das Sprite auf
                        dem Bildschirm befindet
Schuß   lda $c001
        and #$10        Feuerknopf gedrückt?
        beq $Setzten
        rts             Rücksprung ins Hauptprogramm
Setzen  lda $d015
        and #$02
        beq $Einsch.    Schuß einschalten
        rts             Rücksprung ins Hauptprogramm
Einsch. lda $d015
        ora #$02
        sta $d015       Schaltet Schuß ein
        lda $d000
        sta $d002
        lda #$c0
        sta $d003       Positioniert den Schuß
        lda $d010
```

```
        and #$01
        cmp #$01
        beq $Pos         Muß das "Bit 9" für den Schuß ge-
                         setzt werden?
        lda $d010        Ja!
        ora #$02
        sta $d010
Pos     lda $c002
        sta $c003
        lda #$11
        sta $c004        Gibt die Position auf dem Bild-
                         schirm an
        rts              Rücksprung ins Hauptprogramm
```

Teil 5:

Hier wird der Schuß auf dem Bildschirm bewegt.

```
        lda $d015
        and #$02
        cmp #$02         Schuß gesetzt?
        beq $Schuß
        rts              Nein!: Rücksprung
Schuß   ldx #$08
Loop    dec $d003
        dex
        bne $Loop        Bewegt den Schuß
        dec $c004        Verändert Bildschirmposition
        lda $d003
        beq $Lösch       Schuß löschen
        rts              Rücksprung ins Hauptprogramm
Lösch   lda $d015
        and #$fd
        sta $d015        Löscht Sprite
        lda $d010
        and #$fd
```

```
        sta $d010        Löscht "Bit 9"
        rts              Rücksprung ins Hauptprogramm
```

Teil 6:

Dieser Teil verhindert, daß das Raumschiff den Bildschirm verläßt.

```
        lda $d000
        cmp #$18         Linke Grenze des Bildschirms
        bne $Okay
        lda $d010
        and #$01
        cmp #$01         "Bit 9" gesetzt?
        beq $Okay
        lda $c001        Verhindert weitere Bewegung des
        ora #$04         Sprites nach links
        sta $c001        (Ergebnis der Portabfrage wird
                         verändert)
        rts              Rücksprung ins Hauptprogramm
Okay    lda $d000
        cmp #$40         Rechte Grenze des Bildschirms
        beq $Grenze
        rts              Rücksprung ins Hauptprogramm
Grenze  lda $d010
        and #$01
        beq $Nein        "Bit 9" gesetzt?
        lda $c001
        ora #$08
        sta $c001
Nein    rts              Rücksprung ins Hauptprogramm
```

Teil 7:

Hier geschieht die Auswertung, ob ein Vogel aus dem Schwarm getroffen wurde.

In diesem Programmteil wird eine Routine des Betriebssystems genutzt, die Routine zum Setzen des Cursors. Dabei geben die Adressen $c003 und $c004 die Y- bzw. die X-Koordinaten des Cursors an (in diesen beiden Registern steht die Bildschirmposition des Schusses). Durch jsr$fff0 wird der Cursor gesetzt, anschließend kann das Zeichen unter dem (unsichtbaren) Cursor abgefragt und ausgewertet werden. Die Trefferauswertung geschieht also ohne das entsprechende Sprite-Register.

```
        lda $c004
        cmp #$00
        beq $Löschen    Schuß am oberen Bildschirmrand
        nop             Tja, jeder macht mal Fehler
        lda $d015
        and #$02
        cmp #$02        Schuß eingeschaltet?
        beq $Ja
        rts             Rücksprung ins Hauptprogramm.
Ja      ldx $c004
        ldy $c003
        clc             Löscht Carry Flag (wichtig für die
                        Betriebssystemroutine)
        jsr $fff0       Sprung in das Betriebssystem
        ldy $d3
        lda ($d1)       Holt Zeichen unter dem Cursor
        cmp #$44        Treffer?
        beq $Treffer
        rts             Kein Treffer: Rücksprung
Treffer lda #$20
        sta ($d1),y
        iny
        sta ($d1),y
```

```
        dey
        dey
        sta ($d1),y     Löscht untere Hälfte des Vogels
        ldy $c003
        ldx $c004
        dex
        clc
        jsr $fff0       Cursor neu positionieren
        ldy $d3
        lda #$20
        dey
        sta ($d1),y
        iny
        sta ($d1),y
        iny
        sta ($d1),y     Löscht obere Hälfte des Vogels
        inc $c007       Zähler für abgeschossene Vögel
        lda $c007
        cmp #$15        Schwarm komplett gelöscht?
        bne $Nein
        jsr $Teil 9     Neuer Schwarm
Nein    lda #$01
        jsr $teil 10    Aufaddieren des Scores
Löschen lda $d015
        and #$fd
        sta $d015       Löscht Schuß
        lda $d010
        and #$fd
        sta $d010       Löscht "Bit 9" des Schusses
        rts             Rücksprung ins Hauptprogramm
```

Teil 8:

Dieser Teil sorgt für die Bewegung des Schwarms:

```
        inc $c005       Zähler für Teil 8
```

```
        lda $c005
        cmp #$30        Pendeln? (Durch Veränderung des)
                        Wertes wird die Geschwindigkeit
                        beeinflußt
        beq $Ja
        rts             Rücksprung ins Hauptprogramm
        lda #$00
        sta $c005       Initialisiert den Zähler neu
        jmp $Sprung     Entscheidet über links/rechts
                        der Bewegung
Sprung  inc $c006       1. mögliches Sprungziel
        lda $c006
        cmp #$0b        Rechter Bildschirmrand erreicht?
        bne $Nein
        lda #$00
        sta $c006       Initialisiert den Zähler für die
                        Anzahl der Bewegungen
        lda #$Ziel2
        sta $Sprung     Schaltet auf das zweite Sprungziel
                        um (Low-Byte des jmp-Befehls)
Nein    ldx #$ff
Loop    lda $03ff,x
        sta $0400,x
        dex
        bne $Loop       Versetzt Schwarm nach rechts
        rts             Rücksprung ins Hauptprogramm
Sprung  inc $c006       2. mögliches Sprungziel
        lda $c006
        cmp #$0b        Linker Bildschirmrand erreicht?
        bne $Nein2
        lda #$00
        sta $c006       Initialisiert den Zähler für die
                        Anzahl der Bewegungen
        lda #$Ziel1
        sta $Sprung     Schaltet auf das erste Sprungziel
                        um (Siehe oben)
        ldx #$00
Loop2   lda $0400,x
        sta $03ff,x
```

```
        inx
        cpx #$ff
        bne $Loop2      Versetzt Schwarm nach links
        rts             Rücksprung ins Hauptprogramm
```

Teil 9:

Dieser Teil ist eine Unterroutine von Teil 7, er ist nicht direkt vom Hauptprogramm aus erreichbar.

```
        lda #$00
        sta $c007       Initialisiert den Zähler für die
                        Treffer
        ldx #$1d
Loop    lda $c00f,x
        sta $0400,x
        sta $0450,x
        sta $04a0,x
        lda $c02b,x
        sta $0428,x
        sta $0478,x
        sta $04c8,x
        dex             Zeichnet neuen Schwarm.
        bne $Loop
        lda #$00
        sta $c006       Initialisiert Zähler für Teil 8
        lda #$Sprung
        sta $Sprung     Siehe Teil 8 (1. mögliches Ziel)
        rts             Rücksprung zu Teil 7
```

Teil 10:

Auch dieser Teil ist eine Unterroutine zu dem Teil 8. Sie sorgt für die Bewertung der Treffer.

```
        sed                 Setzt BCD Modus
        ldx #$05
Loop    clc
        adc $c050,x
        sta $c050,x
        and #$f0            Übertrag?
        beq $Nein
        lda $#01
        dex
        bpl $Loop
        cld                 Löscht BCD Modus
        ldx #$05
Loop2   lda $c050,x
        and #$0f
        sta $c050,x
        clc
        adc #$30            Errechnet den Bildschirmcode
        sta $07c9,x         Position im Bildschirm
        dex
        bne Loop2
        rts                 Rücksprung in Teil 7
```

Teil 11:

Teil 11 sorgt für unser Hindernis, es schaltet den frei beweglichen Vogel ein.

```
        lda $d015
        and #$04
        cmp #$04        Existiert bereits ein Hindernis?
        bne $Nein
        rts             Es gibt schon einen Vogel
Nein    lda $ROM        Liest ein Zeichen aus dem ROM
        inc $ROM-Low
        sta $d004       X-Position Vogel
        lda #$00
        sta $d005       Y-Position Vogel
        lda $d015
        ora #$04
        sta $d015       Schaltet Vogel ein
        rts             Rücksprung ins Hauptprogramm
```

Teil 12:

Dieser Teil entscheidet, ob der Vogel nach links oder rechts fliegen soll.

```
        inc $c008       Zähler für Bewegung des Vogels
        lda $c008
        cmp #$40        Richtung ändern?
        beq $Event.
        rts             Nein: Rücksprung
Event.  lda #$00
        sta $c008       Zähler initialisieren
        lda $ROM
        inc $ROM-Low    "Zufallswert" holen
        cmp $D000       Mit Position Raumschiff vergleichen
```

```
        bmi $Kleiner
        lda #$01         sta $c009 Nächste Bewegung
nach rechts
        rts              Rücksprung
Kleiner lda #$02
        sta $c009        Nächste Bewegung nach links
        rts              Fertig: Rücksprung
```

Teil 13:

Hier wird der Vogel über den Bildschirm bewegt.

```
        inc $d005        Bewegt Vogel nach unten
        lda $c009
        cmp#$01
        bne $Rechts
        dec $d004        Bewegt Vogel nach links
        lda $d004
        cmp#$ff
        bne $Fertig      Muß "Bit 9" verändert werden?
        lda $d010
        eor #$04
        sta $d010        Verändert "Bit 9" des Vogels
Fertig  rts              Rücksprung ins Hauptprogramm
Rechts  inc $d004        Bewegt Vogel nach Rechts
        bne $Fertig2     Muß "Bit 9" verändert werden?
        lda $d010
        eor #$04
        sta $d010        Verändert "Bit 9"
Fertig2 rts              Rücksprung ins Hauptprogramm
```

Teil 14:

Dieser Teil sorgt dafür, daß ein sich einmal auf dem Bildschirm befindlicher Vogel diesen nicht zu den Seiten hin verlassen kann.

```
        lda $d010
        and #$04
        cmp #$04        Testet "Bit 9" des Vogels
        beq $Gesetzt
        lda $d004
        cmp #$18        Linker Bildschirmrand
        bne $Nein
        lda #$02
        sta $c009       Nächste Bewegung des Sprites nach
                        rechts
Nein    rts             Rücksprung ins Hauptprogramm
Gesetzt lda $d004
        cmp #$40        Rechter Bildschirmrand
        bne $Nein2
        lda #$01
        sta $c009       Nächste Bewegung nach links
Nein2   rts             Rücksprung ins Hauptprogramm
```

Sie sehen, was Erfahrung ausmacht. Diese Routine bewirkt nichts anderes als Teil 6 für unser Raumschiff, die Verwirklichung ist meiner Meinung nach bedeutend eleganter.

Teil 15:

Hier lernt der Vogel das Eierlegen, das zweite Hindernis erscheint auf dem Bildschirm.

```
        lda $d015
```

```
          and #$08
          cmp#$08        Gibt es bereits ein Ei?
          beq $Fallen
          lda $d004      Nein!
          sta $d006      Ei positionieren (X-Koordinate)
          lda $d010
          and #$04
          cmp #$04       "Bit 9" gesetzt?
          bne $Nein
          lda $d010
          ora #$08
          sta $d010      "Bit 9" setzen
Nein      lda $d005
          sta $d007      Y-Koordinate setzen
          lda $d015
          ora #$08
          sta $d015      Ei einschalten
Fallen    inc $d007
          inc $d007      Bewegt Ei um zwei Positionen nach
                         unten
          lda $d007
          and #$fe
          cmp #$00       Ei am unteren Rand?
          beq $Löschen   Nein!
          rts            Rücksprung ins Hauptprogramm
Löschen   lda $d015
          and #$f7
          sta $d015      Ei löschen
          lda $d010
          and #$f7
          sta $d010      "Bit 9" löschen
          rts            Rücksprung ins Hauptprogramm
```

Teil 16:

Im Teil 16 werden die Kollisionen der verschiedenen Sprites untereinander kontrolliert.

```
        lda $d01e      Kollisionsregister Sprite-Sprite
        sta $c00c      Zwischenspeichern
        lda #$00
        sta $d01e      Kollisionregister löschen
        lda $c00c
        and #$01
        cmp #$01       Raumschiff beteiligt?
        bne $Nein
        lda #$01
        sta $d015      Alle Sprites bis auf Raumschiff
                       löschen
        lda #$14
        sta $c002      Mittelposition auf dem Bildschirm
        lda #$b0
        sta $d000      Sprite positionieren
        lda #$00
        sta $d010      Sämtliche "9ten Bits" löschen
        dec $c00a      Zähler Raumschiffe
        bne $Ja
        lda #$01
        sta $c00b      Spiel aus (WAIT-Befehl in Zeile
                       562 des Basic-Programms)
        lda #$31
        sta $0314
        lda #$ea
        sta $0315      Interrupt verändern
        rts            Spiel aus.
Ja      ldx $c00a
        lda #$20
        sta $07dc,x    Verändern der Anzeige auf dem
                       Bildschirm
        rts            Rücksprung ins Hauptprogramm
        lda $c00c
```

```
        cmp #$0c       Kollision Vogel - Ei
        bne $Nein
        rts            Uninteressant: Rücksprung
Nein    cmp #$06       Kollision Schuß - Vogel
        bne $Nein2
        lda $d015
        and #$f9
        sta $d015      Schuß & Vogel löschen
        lda $d010
        and #$f9
        sta $d010      "Bits 9" löschen
        lda #$05
        jsr $Teil10    50 Punkte Belohnung
        rts            Rücksprung ins Hauptprogramm
Nein2   lda $c00c
        cmp #$0a       Kollision Schuß - Ei?
        beq $Ja2
        rts            Rücksprung ins Hauptprogramm
Ja2     lda $d015
        and #$f5
        sta $d015      Schuß & Ei löschen
        lda $d010
        and #$f5
        sta $d010      "Bits 9" löschen
        lda #$05
        jsr $Teil10    50 Punkte Belohnung
        rts            Rücksprung ins Hauptprogramm
```

Hier ist das komplette Maschinensspracheprogramm von Bird.

Auch hier kann man die einzelnen Routinen durch die drei nops als Zwischenraum erkennen.

```
.,  c100  ad 0e dc   lda $dc0e
.,  c103  29 fe      and #$fe
.,  c105  8d 0e dc   sta $dc0e
.,  c108  a9 00      lda #$00
.,  c10a  8d 14 03   sta $0314
.,  c10d  a9 c6      lda #$c6
.,  c10f  8d 15 03   sta $0315
.,  c112  ad 0e dc   lda $dc0e
.,  c115  09 01      ora #$01
.,  c117  8d 0e dc   sta $dc0e
.,  c11a  ea         nop
.,  c11b  ea         nop
.,  c11c  ea         nop
.,  c11d  ee 00 c0   inc $c000
.,  c120  ad 00 c0   lda $c000
.,  c123  c9 08      cmp #$08
.,  c125  f0 01      beq $c128
.,  c127  60         rts
.,  c128  a9 e0      lda #$e0
.,  c12a  8d 02 dc   sta $dc02
.,  c12d  ad 00 dc   lda $dc00
.,  c130  8d 01 c0   sta $c001
.,  c133  a2 ff      ldx #$ff
.,  c135  8e 02 dc   stx $dc02
.,  c138  e8         inx
.,  c139  8e 00 c0   stx $c000
.,  c13c  60         rts
.,  c13d  ea         nop
.,  c13e  ea         nop
.,  c13f  ea         nop
.,  c140  ad 01 c0   lda $c001
.,  c143  29 04      and #$04
.,  c145  d0 1c      bne $c163
```

```
.,  c147  ce 00 d0   dec $d000
.,  c14a  ad 00 d0   lda $d000
.,  c14d  c9 ff      cmp #$ff
.,  c14f  d0 08      bne $c159
.,  c151  ad 10 d0   lda $d010
.,  c154  49 01      eor #$01
.,  c156  8d 10 d0   sta $d010
.,  c159  ad 00 c0   lda $c000
.,  c15c  c9 04      cmp #$04
.,  c15e  d0 03      bne $c163
.,  c160  ce 02 c0   dec $c002
.,  c163  ad 01 c0   lda $c001
.,  c166  29 08      and #$08
.,  c168  d0 17      bne $c181
.,  c16a  ee 00 d0   inc $d000
.,  c16d  d0 08      bne $c177
.,  c16f  ad 10 d0   lda $d010
.,  c172  49 01      eor #$01
.,  c174  8d 10 d0   sta $d010
.,  c177  ad 00 c0   lda $c000
.,  c17a  c9 04      cmp #$04
.,  c17c  d0 03      bne $c181
.,  c17e  ee 02 c0   inc $c002
.,  c181  ad 01 c0   lda $c001
.,  c184  29 10      and #$10
.,  c186  f0 01      beq $c189
.,  c188  60         rts
.,  c189  ad 15 d0   lda $d015
.,  c18c  29 02      and #$02
.,  c18e  f0 01      beq $c191
.,  c190  60         rts
.,  c191  ad 15 d0   lda $d015
.,  c194  09 02      ora #$02
.,  c196  8d 15 d0   sta $d015
.,  c199  ad 00 d0   lda $d000
.,  c19c  8d 02 d0   sta $d002
.,  c19f  a9 c0      lda #$c0
.,  c1a1  8d 03 d0   sta $d003
.,  c1a4  ad 10 d0   lda $d010
```

```
.,  c1a7  29 01     and #$01
.,  c1a9  c9 01     cmp #$01
.,  c1ab  d0 08     bne $c1b5
.,  c1ad  ad 10 d0  lda $d010
.,  c1b0  09 02     ora #$02
.,  c1b2  8d 10 d0  sta $d010
.,  c1b5  ad 02 c0  lda $c002
.,  c1b8  8d 03 c0  sta $c003
.,  c1bb  a9 11     lda #$11
.,  c1bd  8d 04 c0  sta $c004
.,  c1c0  60        rts
.,  c1c1  ea        nop
.,  c1c2  ea        nop
.,  c1c3  ea        nop
.,  c1c4  ad 15 d0  lda $d015
.,  c1c7  29 02     and #$02
.,  c1c9  c9 02     cmp #$02
.,  c1cb  f0 01     beq $c1ce
.,  c1cd  60        rts
.,  c1ce  a2 08     ldx #$08
.,  c1d0  ce 03 d0  dec $d003
.,  c1d3  ca        dex
.,  c1d4  d0 fa     bne $c1d0
.,  c1d6  ce 04 c0  dec $c004
.,  c1d9  ad 03 d0  lda $d003
.,  c1dc  f0 01     beq $c1df
.,  c1de  60        rts
.,  c1df  ad 15 d0  lda $d015
.,  c1e2  29 fd     and #$fd
.,  c1e4  8d 15 d0  sta $d015
.,  c1e7  ad 10 d0  lda $d010
.,  c1ea  29 fd     and #$fd
.,  c1ec  8d 10 d0  sta $d010
.,  c1ef  60        rts
.,  c1f0  ea        nop
.,  c1f1  ea        nop
.,  c1f2  ea        nop
.,  c1f3  ad 00 d0  lda $d000
.,  c1f6  c9 18     cmp #$18
```

```
.,   c1f8  d0 12      bne $c20c
.,   c1fa  ad 10 d0   lda $d010
.,   c1fd  29 01      and #$01
.,   c1ff  c9 01      cmp #$01
.,   c201  f0 09      beq $c20c
.,   c203  ad 01 c0   lda $c001
.,   c206  09 04      ora #$04
.,   c208  8d 01 c0   sta $c001
.,   c20b  60         rts
.,   c20c  ad 00 d0   lda $d000
.,   c20f  c9 40      cmp #$40
.,   c211  f0 01      beq $c214
.,   c213  60         rts
.,   c214  ad 10 d0   lda $d010
.,   c217  29 01      and #$01
.,   c219  f0 08      beq $c223
.,   c21b  ad 01 c0   lda $c001
.,   c21e  09 08      ora #$08
.,   c220  8d 01 c0   sta $c001
.,   c223  60         rts
.,   c224  ea         nop
.,   c225  ea         nop
.,   c226  ea         nop
.,   c227  ad 04 c0   lda $c004
.,   c22a  c9 00      cmp #$00
.,   c22c  f0 53      beq $c281
.,   c22e  ea         nop
.,   c22f  ad 15 d0   lda $d015
.,   c232  29 02      and #$02
.,   c234  c9 02      cmp #$02
.,   c236  f0 01      beq $c239
.,   c238  60         rts
.,   c239  ae 04 c0   ldx $c004
.,   c23c  ac 03 c0   ldy $c003
.,   c23f  18         clc
.,   c240  20 f0 ff   jsr $fff0
.,   c243  a4 d3      ldy $d3
.,   c245  b1 d1      lda ($d1),y
.,   c247  c9 44      cmp #$44
```

```
.,  c249  f0 01     beq $c24c
.,  c24b  60        rts
.,  c24c  a9 20     lda #$20
.,  c24e  91 d1     sta ($d1),y
.,  c250  c8        iny
.,  c251  91 d1     sta ($d1),y
.,  c253  88        dey
.,  c254  88        dey
.,  c255  91 d1     sta ($d1),y
.,  c257  ac 03 c0  ldy $c003
.,  c25a  ae 04 c0  ldx $c004
.,  c25d  ca        dex
.,  c25e  18        clc
.,  c25f  20 f0 ff  jsr $fff0
.,  c262  a4 d3     ldy $d3
.,  c264  a9 20     lda #$20
.,  c266  88        dey
.,  c267  91 d1     sta ($d1),y
.,  c269  c8        iny
.,  c26a  91 d1     sta ($d1),y
.,  c26c  c8        iny
.,  c26d  91 d1     sta ($d1),y
.,  c26f  ee 07 c0  inc $c007
.,  c272  ad 07 c0  lda $c007
.,  c275  c9 15     cmp #$15
.,  c277  d0 03     bne $c27c
.,  c279  20 ed c2  jsr $c2ed
.,  c27c  a9 01     lda #$01
.,  c27e  20 20 c3  jsr $c320
.,  c281  ad 15 d0  lda $d015
.,  c284  29 fd     and #$fd
.,  c286  8d 15 d0  sta $d015
.,  c289  ad 10 d0  lda $d010
.,  c28c  29 fd     and #$fd
.,  c28e  8d 10 d0  sta $d010
.,  c291  60        rts
.,  c292  ea        nop
.,  c293  ea        nop
.,  c294  ea        nop
```

```
.,    c295  ee 05 c0   inc $c005
.,    c298  ad 05 c0   lda $c005
.,    c29b  c9 30      cmp #$30
.,    c29d  f0 01      beq $c2a0
.,    c29f  60         rts
.,    c2a0  a9 00      lda #$00
.,    c2a2  8d 05 c0   sta $c005
.,    c2a5  4c a8 c2   jmp $c2a8
.,    c2a8  ee 06 c0   inc $c006
.,    c2ab  ad 06 c0   lda $c006
.,    c2ae  c9 0b      cmp #$0b
.,    c2b0  d0 0a      bne $c2bc
.,    c2b2  a9 00      lda #$00
.,    c2b4  8d 06 c0   sta $c006
.,    c2b7  a9 c8      lda #$c8
.,    c2b9  8d a6 c2   sta $c2a6
.,    c2bc  a2 ff      ldx #$ff
.,    c2be  bd ff 03   lda $03ff,x
.,    c2c1  9d 00 04   sta $0400,x
.,    c2c4  ca         dex
.,    c2c5  d0 f7      bne $c2be
.,    c2c7  60         rts
.,    c2c8  ee 06 c0   inc $c006
.,    c2cb  ad 06 c0   lda $c006
.,    c2ce  c9 0b      cmp #$0b
.,    c2d0  d0 0a      bne $c2dc
.,    c2d2  a9 00      lda #$00
.,    c2d4  8d 06 c0   sta $c006
.,    c2d7  a9 a8      lda #$a8
.,    c2d9  8d a6 c2   sta $c2a6
.,    c2dc  a2 00      ldx #$00
.,    c2de  bd 00 04   lda $0400,x
.,    c2e1  9d ff 03   sta $03ff,x
.,    c2e4  e8         inx
.,    c2e5  e0 ff      cpx #$ff
.,    c2e7  d0 f5      bne $c2de
.,    c2e9  60         rts
.,    c2ea  ea         nop
.,    c2eb  ea         nop
```

```
.,   c2ec  ea          nop
.,   c2ed  a9 00       lda #$00
.,   c2ef  8d 07 c0    sta $c007
.,   c2f2  a2 1d       ldx #$1d
.,   c2f4  bd 0f c0    lda $c00f,x
.,   c2f7  9d 00 04    sta $0400,x
.,   c2fa  9d 50 04    sta $0450,x
.,   c2fd  9d a0 04    sta $04a0,x
.,   c300  bd 2b c0    lda $c02b,x
.,   c303  9d 28 04    sta $0428,x
.,   c306  9d 78 04    sta $0478,x
.,   c309  9d c8 04    sta $04c8,x
.,   c30c  ca          dex
.,   c30d  d0 e5       bne $c2f4
.,   c30f  a9 00       lda #$00
.,   c311  8d 05 c0    sta $c005
.,   c314  8d 06 c0    sta $c006
.,   c317  a9 a8       lda #$a8
.,   c319  8d a6 c2    sta $c2a6
.,   c31c  60          rts
.,   c31d  ea          nop
.,   c31e  ea          nop
.,   c31f  ea          nop
.,   c320  f8          sed
.,   c321  a2 05       ldx #$05
.,   c323  18          clc
.,   c324  7d 50 c0    adc $c050,x
.,   c327  9d 50 c0    sta $c050,x
.,   c32a  29 f0       and #$f0
.,   c32c  f0 06       beq $c334
.,   c32e  a9 01       lda #$01
.,   c330  ca          dex
.,   c331  10 f0       bpl $c323
.,   c333  d8          cld
.,   c334  a2 05       ldx #$05
.,   c336  bd 50 c0    lda $c050,x
.,   c339  29 0f       and #$0f
.,   c33b  9d 50 c0    sta $c050,x
.,   c33e  18          clc
```

```
.,   c33f  69 30      adc #$30
.,   c341  9d c9 07   sta $07c9,x
.,   c344  ca         dex
.,   c345  d0 ef      bne $c336
.,   c347  60         rts
.,   c348  ea         nop
.,   c349  ea         nop
.,   c34a  ea         nop
.,   c34b  ad 15 d0   lda $d015
.,   c34e  29 04      and #$04
.,   c350  c9 04      cmp #$04
.,   c352  d0 01      bne $c355
.,   c354  60         rts
.,   c355  ad 3a b0   lda $b03a
.,   c358  ee 56 c3   inc $c356
.,   c35b  8d 04 d0   sta $d004
.,   c35e  a9 00      lda #$00
.,   c360  8d 05 d0   sta $d005
.,   c363  ad 15 d0   lda $d015
.,   c366  09 04      ora #$04
.,   c368  8d 15 d0   sta $d015
.,   c36b  60         rts
.,   c36c  ea         nop
.,   c36d  ea         nop
.,   c36e  ea         nop
.,   c36f  ee 08 c0   inc $c008
.,   c372  ad 08 c0   lda $c008
.,   c375  c9 40      cmp #$40
.,   c377  f0 01      beq $c37a
.,   c379  60         rts
.,   c37a  a9 00      lda #$00
.,   c37c  8d 08 c0   sta $c008
.,   c37f  ad 51 a0   lda $a051
.,   c382  ee 80 c3   inc $c380
.,   c385  cd 00 d0   cmp $d000
.,   c388  30 06      bmi $c390
.,   c38a  a9 01      lda #$01
.,   c38c  8d 09 c0   sta $c009
.,   c38f  60         rts
```

```
.,   c390  a9 02     lda #$02
.,   c392  8d 09 c0  sta $c009
.,   c395  60        rts
.,   c396  ea        nop
.,   c397  ea        nop
.,   c398  ea        nop
.,   c399  ee 05 d0  inc $d005
.,   c39c  ad 09 c0  lda $c009
.,   c39f  c9 01     cmp #$01
.,   c3a1  d0 13     bne $c3b6
.,   c3a3  ce 04 d0  dec $d004
.,   c3a6  ad 04 d0  lda $d004
.,   c3a9  c9 ff     cmp #$ff
.,   c3ab  d0 08     bne $c3b5
.,   c3ad  ad 10 d0  lda $d010
.,   c3b0  49 04     eor #$04
.,   c3b2  8d 10 d0  sta $d010
.,   c3b5  60        rts
.,   c3b6  ee 04 d0  inc $d004
.,   c3b9  d0 08     bne $c3c3
.,   c3bb  ad 10 d0  lda $d010
.,   c3be  49 04     eor #$04
.,   c3c0  8d 10 d0  sta $d010
.,   c3c3  60        rts
.,   c3c4  ea        nop
.,   c3c5  ea        nop
.,   c3c6  ea        nop
.,   c3c7  ad 10 d0  lda $d010
.,   c3ca  29 04     and #$04
.,   c3cc  c9 04     cmp #$04
.,   c3ce  f0 0d     beq $c3dd
.,   c3d0  ad 04 d0  lda $d004
.,   c3d3  c9 18     cmp #$18
.,   c3d5  d0 05     bne $c3dc
.,   c3d7  a9 00     lda #$00
.,   c3d9  8d 09 c0  sta $c009
.,   c3dc  60        rts
.,   c3dd  ad 04 d0  lda $d004
.,   c3e0  c9 40     cmp #$40
```

```
.,   c3e2  d0 05       bne $c3e9
.,   c3e4  a9 01       lda #$01
.,   c3e6  8d 09 c0    sta $c009
.,   c3e9  60          rts
.,   c3ea  ea          nop
.,   c3eb  ea          nop
.,   c3ec  ea          nop
.,   c3ed  ad 15 d0    lda $d015
.,   c3f0  29 08       and #$08
.,   c3f2  c9 08       cmp #$08
.,   c3f4  f0 25       beq $c41b
.,   c3f6  ad 04 d0    lda $d004
.,   c3f9  8d 06 d0    sta $d006
.,   c3fc  ad 10 d0    lda $d010
.,   c3ff  29 04       and #$04
.,   c401  c9 04       cmp #$04
.,   c403  d0 08       bne $c40d
.,   c405  ad 10 d0    lda $d010
.,   c408  09 08       ora #$08
.,   c40a  8d 10 d0    sta $d010
.,   c40d  ad 05 d0    lda $d005
.,   c410  8d 07 d0    sta $d007
.,   c413  ad 15 d0    lda $d015
.,   c416  09 08       ora #$08
.,   c418  8d 15 d0    sta $d015
.,   c41b  ee 07 d0    inc $d007
.,   c41e  ee 07 d0    inc $d007
.,   c421  ad 07 d0    lda $d007
.,   c424  29 fe       and #$fe
.,   c426  c9 00       cmp #$00
.,   c428  f0 01       beq $c42b
.,   c42a  60          rts
.,   c42b  ad 15 d0    lda $d015
.,   c42e  29 f7       and #$f7
.,   c430  8d 15 d0    sta $d015
.,   c433  ad 10 d0    lda $d010
.,   c436  29 f7       and #$f7
.,   c438  8d 10 d0    sta $d010
.,   c43b  60          rts
```

```
.,  c43c  ea        nop
.,  c43d  ea        nop
.,  c43e  ea        nop
.,  c43f  ad 1e d0  lda $d01e
.,  c442  8d 0c c0  sta $c00c
.,  c445  a9 00     lda #$00
.,  c447  8d 1e d0  sta $d01e
.,  c44a  ad 0c c0  lda $c00c
.,  c44d  29 01     and #$01
.,  c44f  c9 01     cmp #$01
.,  c451  d0 32     bne $c485
.,  c453  a9 01     lda #$01
.,  c455  8d 15 d0  sta $d015
.,  c458  a9 14     lda #$14
.,  c45a  8d 02 c0  sta $c002
.,  c45d  a9 b0     lda #$b0
.,  c45f  8d 00 d0  sta $d000
.,  c462  a9 00     lda #$00
.,  c464  8d 10 d0  sta $d010
.,  c467  ce 0a c0  dec $c00a
.,  c46a  d0 10     bne $c47c
.,  c46c  a9 01     lda #$01
.,  c46e  8d 0b c0  sta $c00b
.,  c471  a9 31     lda #$31
.,  c473  8d 14 03  sta $0314
.,  c476  a9 ea     lda #$ea
.,  c478  8d 15 03  sta $0315
.,  c47b  60        rts
.,  c47c  ae 0a c0  ldx $c00a
.,  c47f  a9 20     lda #$20
.,  c481  9d dc 07  sta $07dc,x
.,  c484  60        rts
.,  c485  ad 0c c0  lda $c00c
.,  c488  c9 0c     cmp #$0c
.,  c48a  d0 01     bne $c48d
.,  c48c  60        rts
.,  c48d  c9 06     cmp #$06
.,  c48f  d0 16     bne $c4a7
.,  c491  ad 15 d0  lda $d015
```

```
.,   c494   29 f9       and #$f9
.,   c496   8d 15 d0    sta $d015
.,   c499   ad 10 d0    lda $d010
.,   c49c   29 f9       and #$f9
.,   c49e   8d 10 d0    sta $d010
.,   c4a1   a9 05       lda #$05
.,   c4a3   20 20 c3    jsr $c320
.,   c4a6   60          rts
.,   c4a7   ad 0c c0    lda $c00c
.,   c4aa   c9 0a       cmp #$0a
.,   c4ac   f0 01       beq $c4af
.,   c4ae   60          rts
.,   c4af   ad 15 d0    lda $d015
.,   c4b2   29 f5       and #$f5
.,   c4b4   8d 15 d0    sta $d015
.,   c4b7   ad 10 d0    lda $d010
.,   c4ba   29 f5       and #$f5
.,   c4bc   8d 10 d0    sta $d010
.,   c4bf   a9 05       lda #$05
.,   c4c1   20 20 c3    jsr $c320
.,   c4c4   60          rts
.,   c4c5   ea          nop
.,   c4c6   ea          nop
.,   c4c7   ea          nop
```

Dies ist das Hauptprogramm zu dem Spiel "Bird". Es wird vom Interrupt aufgerufen und springt seinerseits die Unterprogramme an.

```
.,  c600  48        pha
.,  c601  98        tya
.,  c602  48        pha
.,  c603  8a        txa
.,  c604  48        pha
.,  c605  20 1d c1  jsr $c11d
.,  c608  20 f3 c1  jsr $c1f3
.,  c60b  20 40 c1  jsr $c140
.,  c60e  20 c4 c1  jsr $c1c4
.,  c611  20 27 c2  jsr $c227
.,  c614  20 95 c2  jsr $c295
.,  c617  20 4b c3  jsr $c34b
.,  c61a  20 6f c3  jsr $c36f
.,  c61d  20 99 c3  jsr $c399
.,  c620  20 c7 c3  jsr $c3c7
.,  c623  20 ed c3  jsr $c3ed
.,  c626  20 3f c4  jsr $c43f
.,  c629  68        pla
.,  c62a  aa        tax
.,  c62b  68        pla
.,  c62c  a8        tay
.,  c62d  68        pla
.,  c62e  4c 31 ea  jmp $ea31
.
.
```

TEIL 4

Nach diesen Beispielprogrammen folgen jetzt einige Tips, wie Sie die vorhandenen Routinen über die Beispiele hinaus noch erweitern oder zweckentfremden können.

15.1. Sprites über den Bildschirm bewegen

Dazu nehmen wir die Steuerungsroutine des Programms "Bird". Sie hat den Vorteil, daß ein Sprite in der X-Richtung über den Bildschirm bewegt werden kann.

Sie können das Sprite auch in Y-Richtung bewegen, wenn Sie in dieser Routine statt des Teiles "Schuß" folgendes einsetzen:

```
        lda $c001
        and #$01
        bne $Unten
        dec $d001
        lda $c000
        cmp #$04
        bne $Unten
        dec $Zeile       Hier muß eine Adresse eingefügt
                         werden, in der die aktuelle Bild-
                         schirmzeile abgelegt werden kann.
Unten   lda $c001
        and #$02
        bne $Schuß
        inc $d001
        lda $c000
        cmp #$04
        bne $Schuß
        inc $Zeile
Schuß
```

So können Sie ein Sprite über den gesamten Bildschirm bewegen. Häufig braucht man dann aber noch eine Routine, die die Form des Sprites je nach Stellung des Joysticks verändert.

Auch das ist kein Problem, eine mögliche Lösung lautet so:

```
         lda $c001
         and #$01        bne $Unten
         lda #$Sprite Block Oben
         sta $07f8       Verändert die Sprite Form
Unten    lda $c001
         and #$02
         bne $Links
         lda #$Sprite Block Unten
         sta $07f8
Links    lda $c001
         and #$04
         bne $Rechts
         lda #$Sprite Block Links
         sta $07f8
Rechts   lda $c001
         and #$08
         bne $Ende
         lda $Sprite Block Rechts
         sta $07f8
Ende     rts
```

Dieser Teil muß natürlich durch das Hauptproramm angesprungen werden.

15.2 Sprite-Editor

Zum Verschieben des Bildschirminhaltes kann man die Routine zur Hintergrundverschiebung im Programm "Street" verwenden. Sie ist im vorliegenden Fall für die Verschiebung von 3 Zeilen gedacht, muß also auf 25 Zeilen erweitert werden.

Ist dies geschehen, so kann der gesamte Bildschirm waagerecht gescrollt werden. Dabei wird der rechte Rand jeweils an den linken hinübergebracht, und umgekehrt. Dafür sorgt die Unterroutine "Retten des Bildschirmrandes".

Hier kann man dann wieder ein wenig tricksen:

Wenn wir nicht den entsprechenden Bildschirmrand auf die andere Seite bringen sondern 25 neue Zeichen, dann kann man einen Hintergrund programmieren, der lediglich durch die Speicherkapazität des Rechners begrenzt wird.

Auf eins müssen wir jedoch bei farbigen Hintergründen achten: Es werden lediglich die Zeichen transferiert, nicht jedoch die entsprechende Farbe. Für sie muß dann also dieselbe Routine für die Farben geschrieben werden.

Auf ähnliche Art ist es dann möglich, den Bildschirm von oben nach unten und anders scrollen zu lasssen.

```
1 p1=49152
2 rem sprite editor
3 data clear, edit, x-expand, y-expand, up, down, left,
right, sprite, store, save, return
4 print"S";:restore
5 fori=0to20
6 print"........................"
7 nexti
8 fori=0to11:reada$
9 poke211,32:poke214,(2*i):sys58640
10 print"r"a$:nexti
11 a=0:b=0
12 poke56322,224:x=127-peek(56320)
13 ifx=0then16
14 x=log(x)/log(2)+1
15 on x gosub18,20,22,24,26
16 c=peek(1024+40*a+b)
17 poke1024+40*a+b,81: fori=1to50: next:
poke1024+40*a+b,c:goto12
18 a=a-1:ifa;0thena=21
19 return
20 a=a+1:ifa:21thena=0
21 return
22 b=b-1:ifb;0thenb=24
23 return
24 b=b+1:ifb:24then gosub29
25 return
26 if peek(1024+40*a+b)=46then c=160:goto28
27 c=46
28 poke1024+40*a+b,c:goto24
29 b=0:d=0:restore
30 poke211,32:poke214,d:sys58640
31 ifd:22then29
32 reada$:printa$
33 if peek(56320)=127or peek(56320)=119 then33
34 if peek(56320);: 111then d=d+2:poke214,d-2:poke211,32:
sys58640: print"r"a$:goto30
35 on(d/2) goto 37,44,47,50,55,61,66,71,82,88,60
36 v=53248:pokev+21,0:goto2
```

```
37 sp=832:s=0:fori=0to20
38 forr=0to2
39 forrr=r*8tor*8+7
40 if peek(1024+i*40+rr)=160 then s=s+2^(8-(rr-8*r)-1)
41 nextrr :pokesp,s:s=0:sp=sp+1: nextr,i
42 v=53248: pokev+21,4: pokev+4,218: pokev+5,52: poke2042,13
43 poke211,32: poke214,d: sys58640: print"r"a$: return
44 if peek(53277)=4 then poke53277,0: goto46
45 poke53277,4
46 poke211,32: poke214,d: sys58640: print"r"a$: return
47 if peek(53271)=4 then poke53271,0: goto49
48 poke53271,4
49 poke211,32: poke214,d: sys58640: print"r"a$: return
50 fori=1to20
51 forr=0to23
52 w=peek(1024+i*40+r): poke1024+(i-1)*40+r,w:
poke1024+i*40+r,46
53 nextr,i
54 poke211,32: poke214,d: sys58640: print"r"a$: return
55 fori=19to0 step-1
56 forr=0to23
57 w=peek(1024+i*40+r): poke1024+(i+1)*40+r,w:
poke1024+i*40+r,46
58 nextr,i
59 poke211,32: poke214,d: sys58640: print"r"a$: return
60 poke211,32: poke214,d: sys58640: print"r"a$: return
61 forr=1to23
62 fori=0to20
63 w=peek(1024+i*40+r): poke1024+i*40+r-1,w:
poke1024+i*40+r,46
64 nexti,r
65 poke211,32: poke214,d: sys58640: print"r"a$: return
66 forr=22to0step-1
67 fori=0to20
68 w=peek(1024+i*40+r): poke1024+i*40+r+1,w:
poke1024+i*40+r,46
69 nexti,r
70 poke211,32: poke214,d: sys58640: print"r"a$: return
71 print"s";: forx1=0to20: fory=0to2
```

```
72 sp=peek(832+x1*3+y)
73 forz=7to 0 step-1
74 if sp and 2^z then sp$(y)=sp$(y)+"r R": goto76
75 sp$(y)=sp$(y)+"."
76 nextz,y
77 sp$=sp$(0)+sp$(1)+sp$(2)
78 poke211,0: poke214,x1: sys58640: printsp$
79 sp$(0)="": sp$(1)="": sp$(2)="": sp$=""
80 nextx1
81 poke211,32: poke214,d: sys58640: print"r"a$: return
82 fori=0to62
83 poke p1+i,peek(832+i)
84 nexti
85 poke p1+63,0
86 p1=p1+64
87 poke211,32: poke214,d: sys58640: print"r"a$: return
88 print"S":pokev+21,0
89 open1,8,1,"0:sprite datas"
90 print#1,chr$(64);: print#1,chr$(3);
91 fori=49152 to p1
92 print#1,chr$(peek(i));
93 nexti: close1: print"S": goto2
```

Mit dem obigen Programm Sprite-Editor steht Ihnen ein Programm zur Verfügung, mit dem Sie auf einfache Art Sprites generieren können.

Tippen Sie das Programm ein und starten Sie es mit RUN

Auf dem Bildschirm erscheint eine 24 * 21 Punktmatrix. Rechts der Matrix erscheint ein einfaches Menue. Der blinkende Cursor innerhalb der Punktmatrix gibt Ihnen den aktuellen Standort innerhalb der Matrix an. Gesteuert werden kann der Cursor mit dem Joystick an Port 2.

Mit der Feuertaste können Sie jetzt beliebige Punkte

innerhalb der Matrix setzen. Stehen Sie auf einem gesetzten Punkt und drücken die Feuertaste, so wird der Punkt gelöscht. Auf diese Art ist ein einfaches Erstellen von Sprites möglich.

Das Menue:

Sie kommen in das Menue, sobald Sie den rechten Rand der Matrix überschreiten. Bewegen Sie den Joystick weiter, so werden nacheinander die Menuepunkte angesprochen. Wollen Sie einen der Punkte ansprechen, so müssen Sie auf die Feuertaste drücken.

Clear:
Löscht den gesamten Bildschirm.

Edit:
Formt den Bildschirminhalt zu einem Sprite um.

X-Expand:
Vergrößert ein Sprite in X-Richtung. Um das Sprite wieder auf normale Größe zu bringen, müssen Sie diesen Menuepunkt erneut ansprechen.

Y-Expand:
Vergrößert ein Sprite in Y-Richtung. Auch hier verkleinert ein zweites Ansprechen das Sprite wieder.

Up:
Dieser Menuepunkt bietet eine Hilfe, wenn Sie sich in der Größe Ihres Sprite vertan haben. Up schiebt die Matrix um eine Position nach oben, die oberste Zeile geht dabei verloren.

Down:
Verschiebt die Matrix nach unten.

Left:
Verschiebt die Matrix nach links.

Right:
Verschiebt die Matrix nach rechts.

Sprite:
Die Funktion "Sprite" holt die Informationen aus dem Sprite-Block 13 und bringt sie in Form der Punktmatrix auf den Bildschirm. Gedacht ist sie für den Fall, daß Sie an einem Sprite so sehr herumgedoktert haben, daß es Ihnen nicht mehr gefällt, und Sie wieder die alte Vorlage auf dem Bildschirm haben wollen.

Store:
Speichert die Informationen aus dem Sprite Block 13 ab 49152 um. Immer, wenn Sie ein weiteres Sprite definieren wollen, das alte aber auch behalten wollen, sollten Sie die Funktion Store ansprechen.

Save:
Speichert die definierten und mit Store gespeicherten Sprites auf der Diskette unter den Namen Sprite Datas ab. Mit LOAD,8,1 laden Sie die Daten ab 832 (Sprite Block 13) wieder in den Speicher. Sollen die Daten an eine andere Stelle geladen werden, so müssen Sie vor dem Starten des Programms die Zeile 90 entsprechend ändern.

Return:
Kehrt in die Punktmatrix zurück.

Das Programm ist komplett in Basic geschrieben, Sie sollten also nicht allzuviel von der Geschwindigkeit erwarten. Schneller als mit Papier und Bleistift geht es dennoch allemal.

15.3. DATA-Erzeuger

```
63000 INPUT "▯TARTADRESSE                    "
;AD:AD=AD-1
63010 INPUT "█NDADRESSE                      "
;SE
63020 INPUT "█IE VIELE S PRO EILE ";DA:I
F DA<1 OR DA>16 THEN 63020
63030 POKE2,DA
63040 POKE1020,SE-INT(SE/256)*256:POKE10
21,SE/256
63050 INPUT"█B EILE                     ";Z
%:Z%=Z%-1
63100 Z%=Z%+1
63110 Z$=STR$(Z%)+"D"
63120 J=J+1:AD=AD+1:P%=PEEK(AD):PF%=PF%+
P%
63130 Z$=Z$+STR$(P%)+","
63140 IFJ<PEEK(2)ANDAD<SETHEN63120
63150 Z$=Z$+STR$(PF%)
63160 PRINT"▯██";Z$:PRINT"GOTO 63300"
63170 POKE251,Z%AND255:POKE252,Z%/256
63180 POKE253,AD-INT(AD/256)*256:POKE254
,AD/256
63200 POKE631,13:PRINT"█";
63210 IFAD<SETHENPOKE632,13:POKE198,2:EN
D
63220 POKE198,1:END
63300 Z%=PEEK(251)+256*PEEK(252)
63310 AD=PEEK(253)+256*PEEK(254)
63320 SE=PEEK(1020)+256*PEEK(1021)
63330 GOTO 63100
```

Dieses Programm erzeugt DATA-Zeilen von einem beliebigen Speicherbereich.

15.4. Dez in N

```
30 CLR
40 PRINT"DEZ->N=N UND N->DEZ=D"
50 GET A$
60 IF A$="N" GOTO 1000
70 IF A$="D" GOTO 3000
80 GOTO 50
1000 INPUT "ZAHLENSYSTEM:";N
1010 INPUT"X=";X
1020 A=LOG (X)/LOG(N)
1030 A=INT (A)
1040 FOR B=A TO 0 STEP -1
1050 IF(X-INT (N↑B+.5))<0 THEN 2020
1060 D=INT (X/INT (N↑B+.5))
1070 IF D>9 GOTO 2000
1071 H$=CHR$(D+48)
1072 E$=E$+H$
1075 X=X-D*INT (N↑B+.5)
1090 NEXT
1100 PRINT E$
1110 GET A$:IF A$="" GOTO 1110
1120 GOTO 30
2000 H$=CHR$(D+55)
2010 GOTO 1072
2020 E$=E$+"0":GOTO 1090
2030 E$=MID$(X$,A-B,1)
2040 IF E$="0" THEN X=0:GOTO 3070
2050 X=ASC(E$)-55
2060 GOTO 3060
```

```
3000 INPUT "ZAHLENSYSTEM:";N
3010 INPUT "X=";X$
3020 A=LEN(X$)
3030 FOR B=0 TO A-1
3040 X=VAL (MID$(X$,A-B,1))
3050 IF X=0 GOTO 2030
3060 E=E+X*INT (N↑B+.5)
3070 NEXT
3080 PRINTE
3090 GOTO 1110
```

Dieses Programm wandelt beliebige Zahlen in andere Zahlensysteme um.



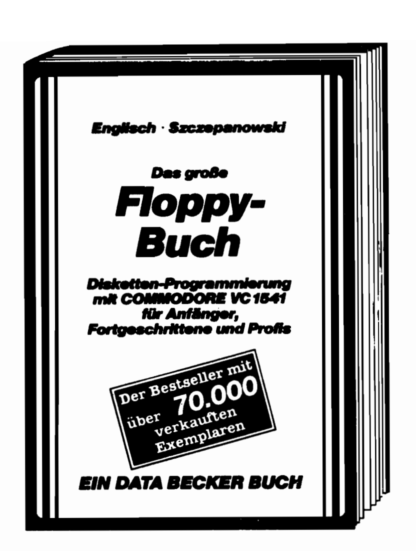
Die Herausforderung für jeden ernsthaften C-64-Anwender! Alles über Technik, Betriebssystem und Programmierung des Commodore 64: kommentiertes ROM-Listing, Speicherbelegungspläne, SID- und VIC-Chip, Ein-/Ausgabesteuerung, Timer und Echtzeituhr, BASIC-Interpreter, mathematische Routinen, IEC-Bus, RS-232, Zeropage und Originalschaltpläne. Das Standardwerk zur Hardware des C-64!

Angerhausen/Brückmann/ Englisch/Gerits
64 Intern
352 Seiten, 2 Schaltpläne, DM 69,–
ISBN 3-89011-000-2

Wester
Das Betriebssystem des COMMODORE 64
EIN DATA BECKER BUCH

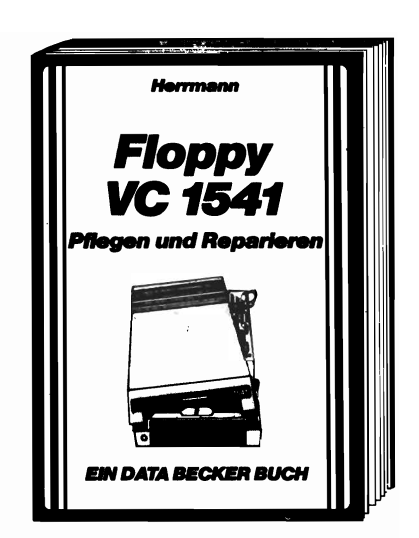
79 (!) Routinen des Betriebssystems enthält dieses Buch. Z. B.: Eingabe einer Zeile per Tastatur, String ausgeben, Ausgabe eines ASCII-Zeichens, beliebigen Ausdruck holen, Multiplikation/Division und Cursor setzen/holen. Startadresse, Einsprungbedingungen, Akku, Register und Flags werden jeweils beschrieben. Ein unverzichtbares Hilfsmittel für jeden Maschinenspracheprogrammierer!

Wester
Das Betriebssystem des Commodore 64
177 Seiten, DM 29,–
ISBN 3-89011-136-X



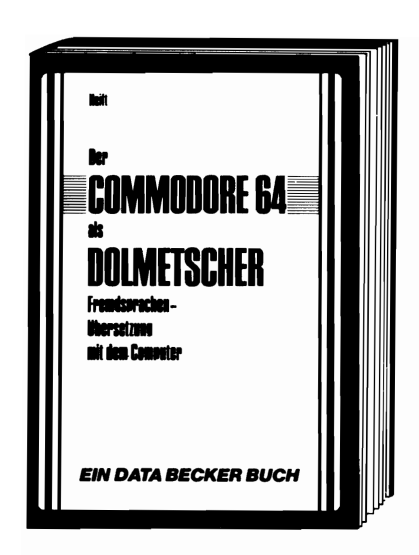
Das Standardwerk zur Programmierung der Floppy 1541! Neben den Systembefehlen, den Fehlermeldungen und dem kommentierten DOS erfahren Sie alles über sequentielle und relative Dateiverwaltung. Dazu viele Programme: Scratch-Schutz, Diskname und ID verändern, Spooling, Overlay, Merge und einen komfortablen Diskmonitor. Dieses Buch zeigt, daß die Floppy nur zum Speichern viel zu schade ist.

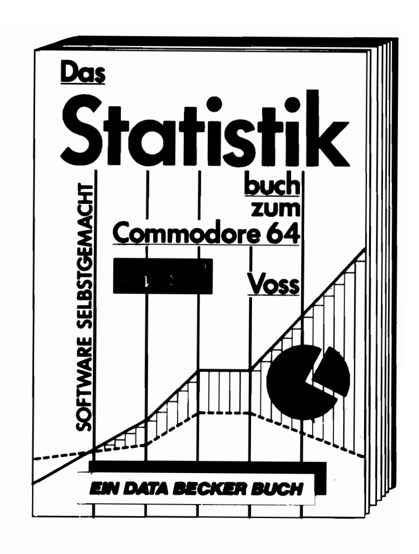
Englisch/Szczepanowski
Das große Floppy-Buch
481 Seiten, DM 49,–
ISBN 3-89011-005-3



Selbsthilfe spart Zeit, Ärger und Geld! Gerade die Floppyjustage oder Reparaturen der Platine sind oft mit einfachen Mitteln zu bewältigen. Anleitungen zur Behebung der meisten Störfälle, Ersatzteillisten und eine Einführung in Mechanik und Elektronik des Laufwerks machen dieses Buch in jeder Beziehung zu einem „preiswerten" und effektiven Buch!

Hermann
VC-1541 Pflegen und
Reparieren
220 Seiten, DM 49,–
ISBN 3-89011-079-7





Der C-64 als Sprachgenie! Aber nicht nur Computersprachen. Wie wär's mit Englisch, Französisch oder Latein? Benutzen Sie Ihren Rechner doch als Dolmetscher. Dieses Buch enthält ein komplettes Programmentwicklungssystem zur Erstellung eines Fremdwortlexikons und eines Textübersetzungssystems mit Grob- und Feinübersetzung. Begeben Sie sich auf ein völlig neues und faszinierendes Gebiet! Leichtverständlich!

Heift
Der Commodore 64
als Dolmetscher
274 Seiten, DM 49,–
ISBN 3-89011-069-X

Statistik auf dem C-64! Ein Lehr- und Arbeitsbuch, über die Grundlagen der Statistik mit zahlreichen Programmen: Häufigkeitstabellen, Mittelwerte und Streuungen, Regressions- und Korrelationsberechnungen, Zeitreihenstatistik, Hochrechnungen u. v. m. BASIC-Programme leicht an eigene Anwendungen anpaßbar. Das ist Software zum Selbermachen!

Voß
Das Statistikbuch zum C-64
448 Seiten, DM 49,–
ISBN 3-89011-102-5



Schauen Sie ins Innere Ihres Rechners! Leichtverständlich wird in diesem Buch der Umgang mit PEEK- und POKE-Befehlen erklärt. Außerdem Grundlegendes zum Aufbau des C-64: Betriebssystem, Interpreter, Zeropage, Pointer und Stacks, Charakter-Generator, Sprite-Register und vieles mehr. Mit einer ersten Einführung in die Maschinensprache und etlichen Beispielprogrammen.

Liesert
Peeks & Pokes zum
Commodore 64
177 Seiten, DM 29,–
ISBN 3-89011-032-0



Dem interessierten Anfänger werden hier die weitverbreiteten Assembler Profimat, MAE 64 und T.EX.AS. ausführlich anhand von Übungen und Beispielen erklärt. Zugleich eine konsequente Einführung in die Maschinensprache. Ein umfassender und übersichtlicher Anhang mit Erläuterungen aller Begriffe sowie ein reichhaltiges Stichwortverzeichnis ergänzen dieses Trainingsbuch optimal.

Schmidt
Assembler Trainingsbuch
Profimat, MAE 64 T.EX.AS.
264 Seiten, DM 39,–
ISBN 3-89011-071-1



Der Bestseller zur Grafikprogrammierung des C-64. Lernen Sie den VIC-Chip kennen! Bringt alles über Farben, Multicolor und Hi-Res-Grafik, Sprites, Hardcopys, 3-D-Grafik CAD, Spielegrafik und Statistik. Mit vielen Beispiel- und Hilfsprogrammen zum Abtippen. Mit diesem Buch werden Sie zum Bildschirmkünstler!

Plenge
Das Grafikbuch zum Commodore 64
295 Seiten, DM 39,–
ISBN 3-89011-011-8

Dullin · Strassenburg
Das große
EPSON
Druckerbuch
EIN DATA BECKER BUCH

EPSON-Drucker sind Standard auf dem Druckermarkt. Dieses Buch macht Schluß mit allen Anschluß- und Steuerproblemen! Von der Beschreibung der Mechanik und Elektronik über die technischen Daten der verschiedenen Typen bis zur Kommunikation mit dem Rechner, der Schriftbildsteuerung und der Formular- und Grafikausgabe ist alles ausführlich und leicht verständlich erklärt. Nutzen sie die Möglichkeiten Ihres EPSON-Druckers!

Dullin/Straßenburg
Das große
EPSON-Drucker-Buch
ca. 250 Seiten, DM 49,–
Erscheint Anfang Dezember
ISBN 3-89011-139-4

DAS STEHT DRIN:
Mit diesem Buch wird das Programmieren auch anspruchsvollerer Videospiele auf dem Commodore 64 wirklich leichtgemacht. Neben einer umfassenden Einführung in die Grundlagen der Spieleprogrammierung enthält es ausführlich erläuterte Beispielprogramme sowie viele nützliche Routinen zur Entwicklung eigener Spiele.

Aus dem Inhalt:

- Sprites
- Sprite-Editor
- Hochauflösende Grafik
- Soundprogrammierung
- Data-Erzeuger
- Basic kontra Maschinensprache
- Umwandlungsprogramm für Zahlensysteme
- Pink Panther
- Labyrinth
- Bird
- 3-D-Autorennen

und vieles mehr.

UND GESCHRIEBEN HAT DIESES BUCH:
Rüdiger Linden beschäftigt sich engagiert mit der Entwicklung von Videospielen und programmiert sie selbst seit langem auf seinem Commodore 64.

ISBN 3-89011-087-8